VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Saney Vasudha Sree(1BF24CS271)

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 **Saney Vasudha Sree(1BF24CS271)** who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 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.

Prof .Anusha S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

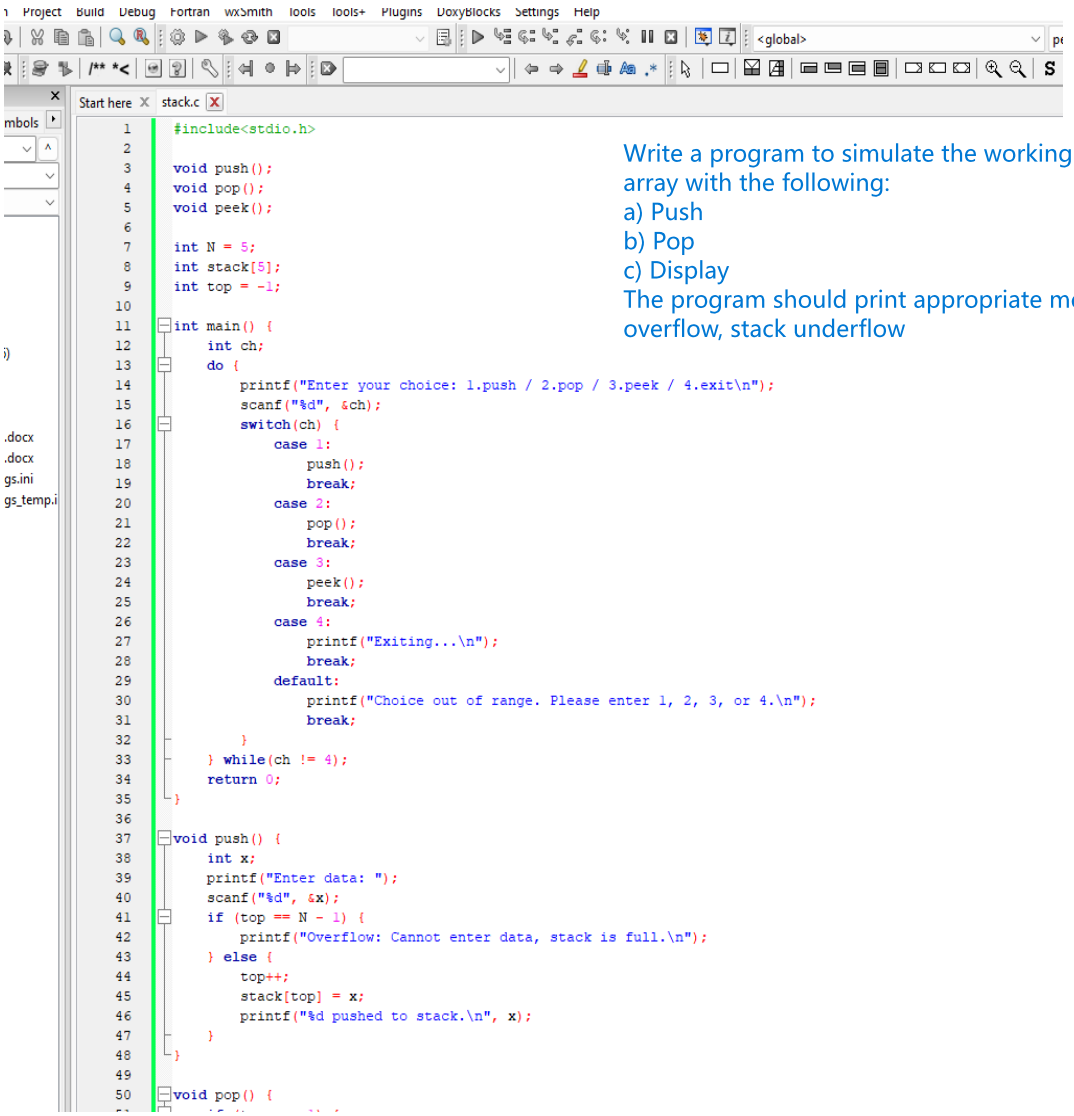
Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

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

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

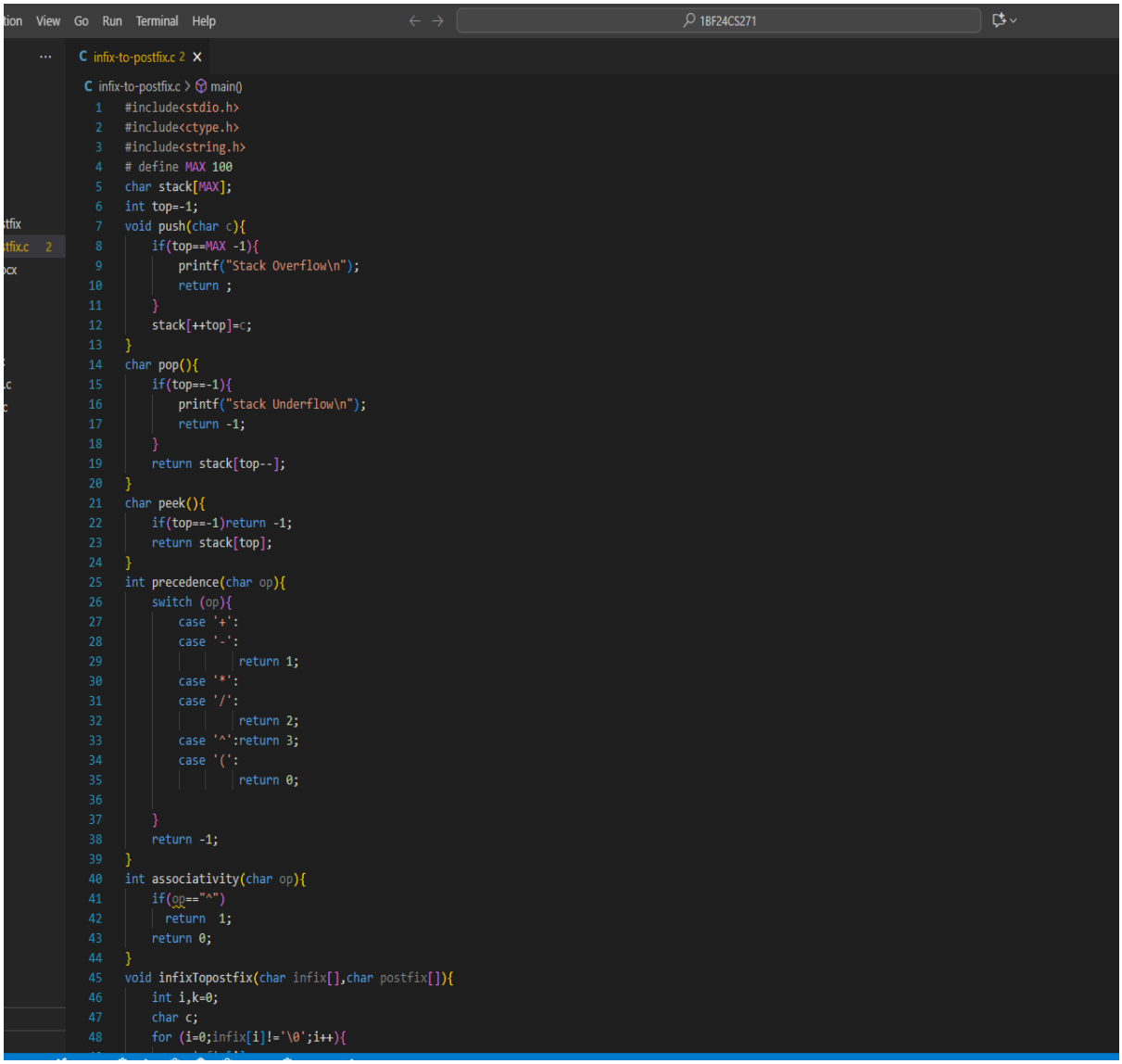


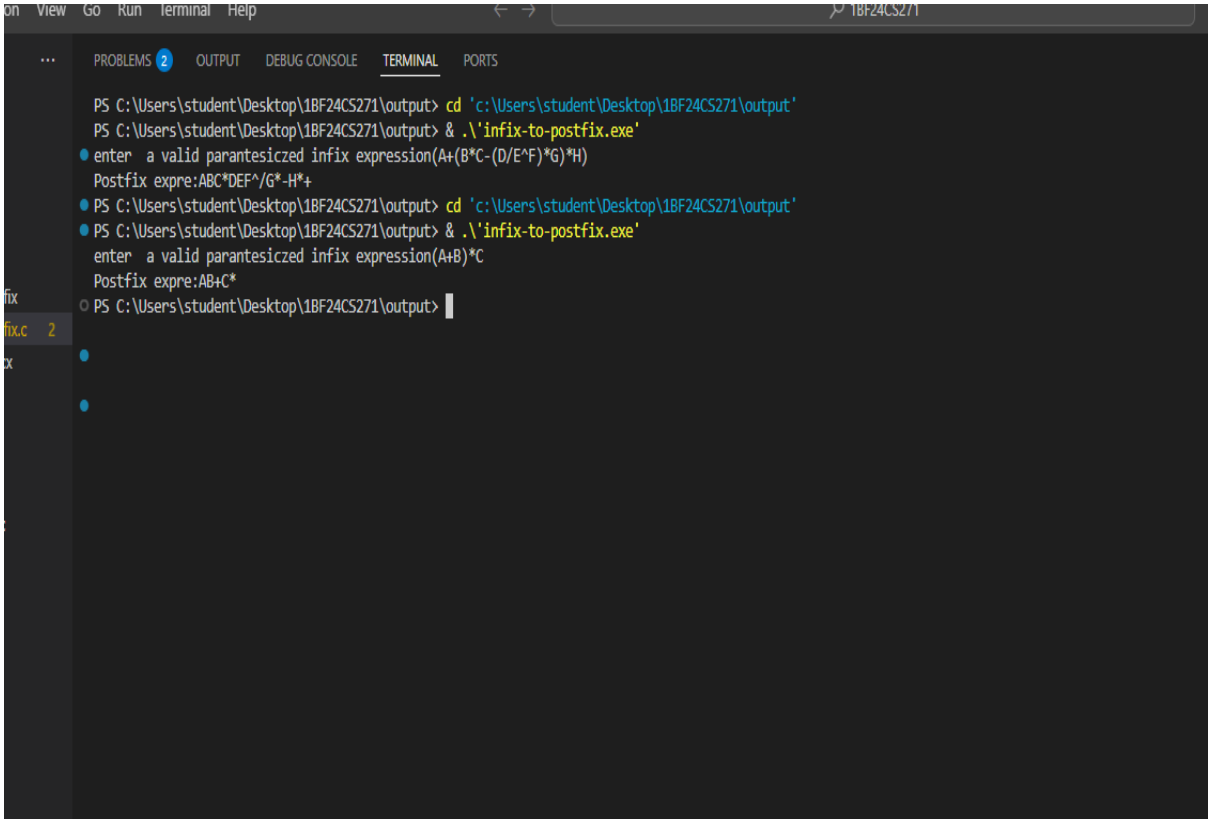

```c
#include<stdio.h>

void push();
void pop();
void peek();

int N = 5;
int stack[5];
int top = -1;

int main() {
    int ch;
    do {
        printf("Enter your choice: 1.push / 2.pop / 3.peek / 4.exit\n");
        scanf("%d", &ch);
        switch(ch) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                peek();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Choice out of range. Please enter 1, 2, 3, or 4.\n");
                break;
        }
    } while(ch != 4);
    return 0;
}

void push() {
    int x;
    printf("Enter data: ");
    scanf("%d", &x);
    if (top == N - 1) {
        printf("Overflow: Cannot enter data, stack is full.\n");
    } else {
        top++;
        stack[top] = x;
        printf("%d pushed to stack.\n", x);
    }
}

void pop() {
```

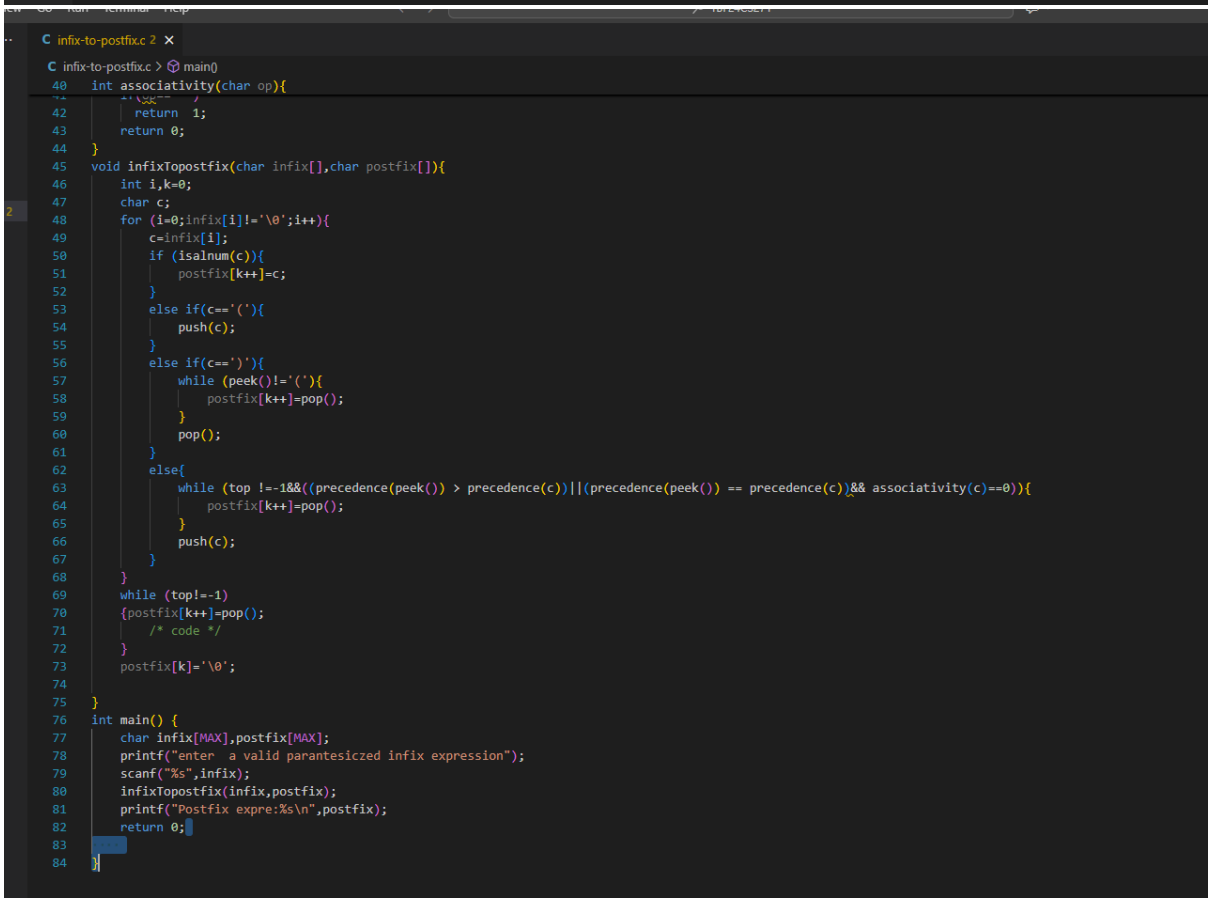
```
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
1
Enter data: 10
10 pushed to stack.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
1
Enter data: 20
20 pushed to stack.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
1
Enter data: 30
30 pushed to stack.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
1
Enter data: 40
40 pushed to stack.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
1
Enter data: 50
50 pushed to stack.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
1
Enter data: 60
Overflow: Cannot enter data, stack is full.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
3
Top item: 50
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
2
Popped item: 50
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
2
Popped item: 40
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
2
Popped item: 30
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
2
Popped item: 20
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
2
Popped item: 10
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
2
Underflow: Stack is empty.
Enter your choice: 1.push / 2.pop / 3.peek / 4.exit
4
Exiting...

Process returned 0 (0x0)   execution time : 1259.643 s
Press any key to continue.
```

```
}

void pop() {
    if (top == -1) {
        printf("Underflow: Stack is empty.\n");
    } else {
        int item = stack[top];
        top--;
        printf("Popped item: %d\n", item);
    }
}

void peek() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Top item: %d\n", stack[top]);
    }
}
```

lab 2
WAP to convert a given valid parenthesized infix
arithmetic expression to postfix expression. The expression consists of single character c
binary operators + (plus), - (minus), * (multiply) and /(divide)




View Go Run Terminal Help
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\student\Desktop\1BF24CS271\output> cd 'c:\Users\student\Desktop\1BF24CS271\output'
PS C:\Users\student\Desktop\1BF24CS271\output> & .\'infix-to-postfix.exe'
enter a valid parantesiczed infix expression(A+(B*C-(D/E^F)*G)*H)
Postfix expre:ABC*DEF^/G*-H*+
PS C:\Users\student\Desktop\1BF24CS271\output> cd 'c:\Users\student\Desktop\1BF24CS271\output'
PS C:\Users\student\Desktop\1BF24CS271\output> & .\'infix-to-postfix.exe'
enter a valid parantesiczed infix expression(A+B)*C
Postfix expre:AB+C*
PS C:\Users\student\Desktop\1BF24CS271\output>


infix-to-postfix.c 2
infix-to-postfix.c > main()
int associativity(char op){
return 1;
return 0;
}
void infixTopostfix(char infix[],char postfix[]){
int i,k=0;
char c;
for (i=0;infix[i]!='\0';i++){
c=infix[i];
if (isalnum(c)){
postfix[k++]=c;
}
else if(c=='('){
push(c);
}
else if(c==')'){
while (peek()!='('){
postfix[k++]=pop();
}
pop();
}
else{
while (top !=-1&&((precedence(peek()) > precedence(c))||(precedence(peek()) == precedence(c))&& associativity(c)==0)){
postfix[k++]=pop();
}
push(c);
}
}
while (top!=-1)
{postfix[k++]=pop();
/* code */
}
postfix[k]='\0';
}
int main() {
char infix[MAX],postfix[MAX];
printf("enter a valid parantesiczed infix expression");
scanf("%s",infix);
infixTopostfix(infix,postfix);
printf("Postfix expre:%s\n",postfix);
return 0;
}

Lab program 3

a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display
The programshould print appropriate messagesfor queue empty and queue overflow conditions

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

#define N 5
int queue[N];
int front = -1;
int rear = -1;

void enque(int x)
{
   if (rear == N - 1)
   {
      printf("Queue Overflow\n");
   }
   else if (front == -1 && rear == -1)
   {
      front = rear = 0;
      queue[rear] = x;
      printf("inserted element=%d",x);
   }
   else
   {
      rear++;
      queue[rear] = x;
      printf("inserted element=%d",x);
   }
}

void deque()
{
```

```
    if (front == -1 && rear == -1)
    {
        printf("Queue is empty\n");
    }
    else if (front == rear)
    {
        printf("Deleted element = %d\n", queue[front]);
        front = rear = -1;
    }
    else
    {
        printf("Deleted element = %d\n", queue[front]);
        front++;
    }
}

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

```
}

void peek()
{
   if (front == -1 && rear == -1)
   {
      printf("Queue is empty\n");
   }
   else
   {
      printf("Front element = %d\n", queue[front]);
   }
}

int main()
{
   int ch, x;
   while (1)
   {
      printf("\nEnter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         printf("Enter element to insert: ");
         scanf("%d", &x);
         enque(x);
         break;
      case 2:
         deque();
         break;
```

```
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            exit(0);
        default:
            printf("Choice out of range\n");
        }
    }
    return 0;
}
```

```
inserted element=20
Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 1
Enter element to insert: 30
inserted element=30
Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 3
Queue elements: 10 20 30

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 4
Front element = 10

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 2
Deleted element = 10

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 2
Deleted element = 20

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 2
Deleted element = 30

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 2
Queue is empty

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 3
Queue is empty

Enter your choice:1. Enqueue 2. Dequeue 3. Display 4. Peek 5. Exit 5

Process returned 0 (0x0)   execution time : 39.150 s
Press any key to continue.
```

lab 3b

circularqueue.c

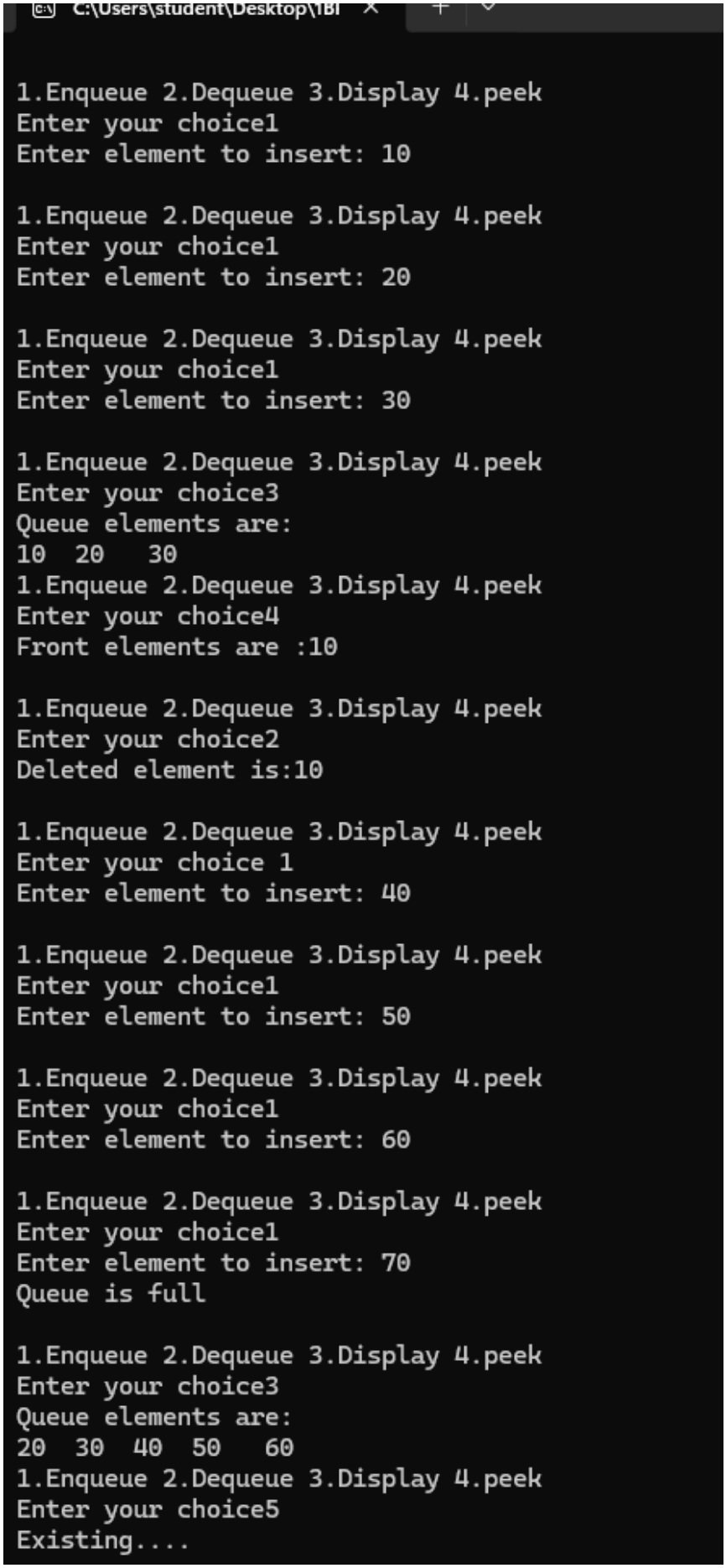
```c
#include<stdio.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;
void enqueue(int x)
{
    if(front==-1 && rear==-1)
    {
        front=rear=0;
        queue[rear]=x;
    }
    else if((rear+1)%N==front)
    {
        printf("Queue is full\n");
    }
    else{
        rear=(rear+1)%N;
        queue[rear]=x;
    }
}
void dequeue()
{

    if(front==-1 && rear==-1)
    {
        printf("Queue is empty\n");

    }
    else if(front==rear)
    {

        printf("Deleted element is :%d\n",queue[front]);

    }
    else
    {
```

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

```
    }
    else
    {
        printf("Deleted element is:%d\n",queue[front]);
        front=(front+1)%N;
    }

void display()

    if(front==-1 && rear==-1)
    {
        printf("Queue is empty\n");
    }
    else{
            int i;
        printf("Queue elements are:\n");
        for ( i=front;i!=rear;i=(i+1)%N)
        {
            printf("%d  ",queue[i]);
        }
        printf(" %d",queue[i]);
    }

void peek()

    if(front==-1 &&rear==-1)
    {
        printf("Queue is empty\n");
    }
    else
    {
        printf("Front elements are :%d\n",queue[front]);
    }

int main()
```

circularqueue.c

```c
}
void peek()
{
    if(front==-1 &&rear==-1)
    {
        printf("Queue is empty\n");
    }
    else
    {
        printf("Front elements are :%d\n",queue[front]);
    }
}
int main()
{
    int choice,x;
    while(1)
    {
        printf("\n1.Enqueue 2.Dequeue 3.Display 4.peek\n");
        printf("Enter your choice");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("Enter element to insert: ");
                   scanf("%d",&x);
                   enqueue(x);
                   break;
            case 2: dequeue();break;
            case 3:display();break;
            case 4:peek();break;
            case 5:printf("Existing....\n");exit(0);break;
            default:printf("INvalid choice");
        }
    }
    return 0;
}
```


C:\Users\student\Desktop\IBI
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice1
Enter element to insert: 10
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice1
Enter element to insert: 20
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice1
Enter element to insert: 30
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice3
Queue elements are:
10 20 30
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice4
Front elements are :10
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice2
Deleted element is:10
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice 1
Enter element to insert: 40
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice1
Enter element to insert: 50
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice1
Enter element to insert: 60
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice1
Enter element to insert: 70
Queue is full
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice3
Queue elements are:
20 30 40 50 60
1.Enqueue 2.Dequeue 3.Display 4.peek
Enter your choice5
Existing....

lab 4a

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

linkedlistinsertion.c | circularqueue.c

```c
#include<stdio.h>
#include<stdlib.h>
struct Node{
int data;
struct Node *next;
};
struct Node *head=NULL;
void createList(int n){
struct Node *newNode,*temp;
int data,i;
if(n<=0){printf("Number of nodes should be greater than 0\n");
return ;}
for(i=1;i<=n;i++)
{
    newNode=(struct Node *)malloc(sizeof(struct Node));
    if(newNode==NULL){
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter data for node %d: ",i);
    scanf("%d",&data);
    newNode ->data=data;
    newNode->next=NULL;
    if(head==NULL){
        head=newNode;
    }
    else{
        temp->next=newNode;
    }
    temp=newNode;
    }
    printf("\nLinked list created successfully\n");
}
void insertAtBeginning(int data){
struct Node *newNode=(struct Node*)malloc (sizeof(struct Node));
newNode->data=data;
newNode->next=head;
```

```
newNode->next=head;
head=newNode;
printf("Node inserted at the beginning\n");
}
void insertionAtEnd(int data){
struct Node *newNode=(struct Node*)malloc (sizeof(struct Node));
newNode->data=data;
newNode->next=NULL;
if(head==NULL){
    head=newNode;
}
else{
    struct Node *temp=head;
    while(temp->next!=NULL)
        temp=temp->next;
    temp->next=newNode;
}
printf("Node inserted at the end\n");
}
void insertAtPosition(int data,int pos){
int i;
struct Node *newNode,*temp=head;
if(pos<1){
    printf("Invalid position\n");
    return;
}
if(pos==1){
    insertAtBeginning(data);
    return;
}
newNode=(struct Node*)malloc(sizeof(struct Node));
newNode->data=data;
for(i=1;i<pos-1 && temp!=NULL;i++)
    temp=temp->next;
if(temp==NULL){
    printf("Position out of range\n");
    free(newNode);
```

```
    free(newNode);
}else{
newNode->next=temp->next;
temp->next=newNode;
printf("Node inserted at position %d\n",pos);
}

}
void displayList(){
struct Node *temp=head;
if(head==NULL){
    printf("List is empty\n");
}
printf("\nLinked List: ");
while (temp!=NULL){
    printf("%d -> ",temp->data);
    temp=temp->next;
}
printf("NULL\n");
}
int main(){
int choice,n,data,pos;
while(1){
    printf("\n---SIngly Linked List Operations---\n");
    printf("1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
    case 1:printf("Enter num of nodes");
           scanf("%d" ,&n);
           createList(n);
           break;
    case 2:printf("Enter data to insert :");
            scanf("%d",&data);
            insertAtBeginning(data);
            break;
    case 3:printf("enter data and position: ");
```

```c
int main(){
 int choice,n,data,pos;
while(1){
     printf("\n---SIngly Linked List Operations---\n");
     printf("1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit\n");
     printf("Enter your choice: ");
     scanf("%d",&choice);
     switch(choice){
     case 1:printf("Enter num of nodes");
            scanf("%d" ,&n);
            createList(n);
            break;
     case 2:printf("Enter data to insert :");
             scanf("%d",&data);
             insertAtBeginning(data);
             break;
     case 3:printf("enter data and position: ");
              scanf("%d %d",&data,&pos);
              insertAtPosition(data,pos);
              break;
     case 4:
         printf("Enter data to insert:");
         scanf("%d",&data);
         insertionAtEnd(data);break;
     case 5:
         displayList();
         break;
     case 6:printf("exiting...");
             exit(0);

     default:printf("Invalid choice Try again\n");
     }
}
return 0;
}
```

```
---SIngly Linked List Operations---
1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit
Enter your choice: 1
Enter num of nodes5
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 40
Enter data for node 5: 50

Linked list created successfully

---SIngly Linked List Operations---
1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit
Enter your choice: 2
Enter data to insert :60
Node inserted at the beginning

---SIngly Linked List Operations---
1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit
Enter your choice: 3
enter data and position: 35
4
Node inserted at position 4

---SIngly Linked List Operations---
1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit
Enter your choice: 4
Enter data to insert:55
Node inserted at the end

---SIngly Linked List Operations---
1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit
Enter your choice: 5

Linked List: 60 -> 10 -> 20 -> 35 -> 30 -> 40 -> 50 -> 55 -> NULL

---SIngly Linked List Operations---
1.Create Linked List 2.Insert at Beginning 3.Insert at any position 4.insert at end 5.display 6.exit
Enter your choice: 6
exiting...
Process returned 0 (0x0)   execution time : 88.720 s
Press any key to continue.
```

lab 4b leetcode

leetcode.com/problems/middle-of-the-linked-list/

Problem List | Submit | Premium

Description | Editorial | Solutions | Submissions

876. Middle of the Linked List

Solved

Easy | Topics | Companies

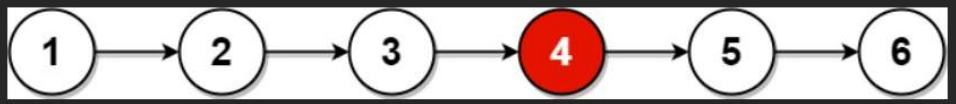
Given the `head` of a singly linked list, return *the middle node of the linked list.*

If there are two middle nodes, return **the second middle** node.

Example 1:


```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:




```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return
```

13K | 246 | 103 Online

Code

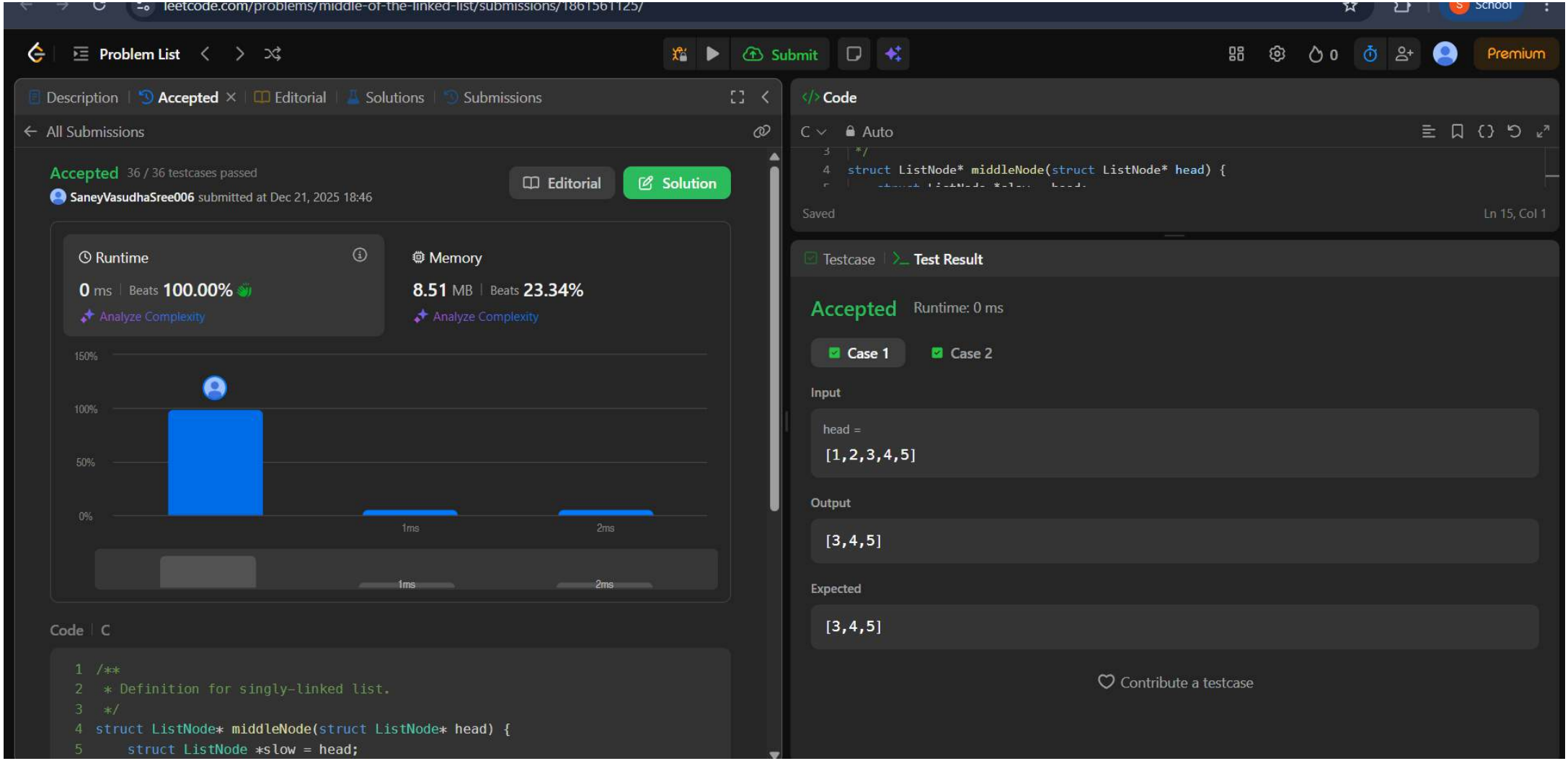
C | Auto

```c
/**
 * Definition for singly-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;
}
```

Saved | Ln 15, Col 1

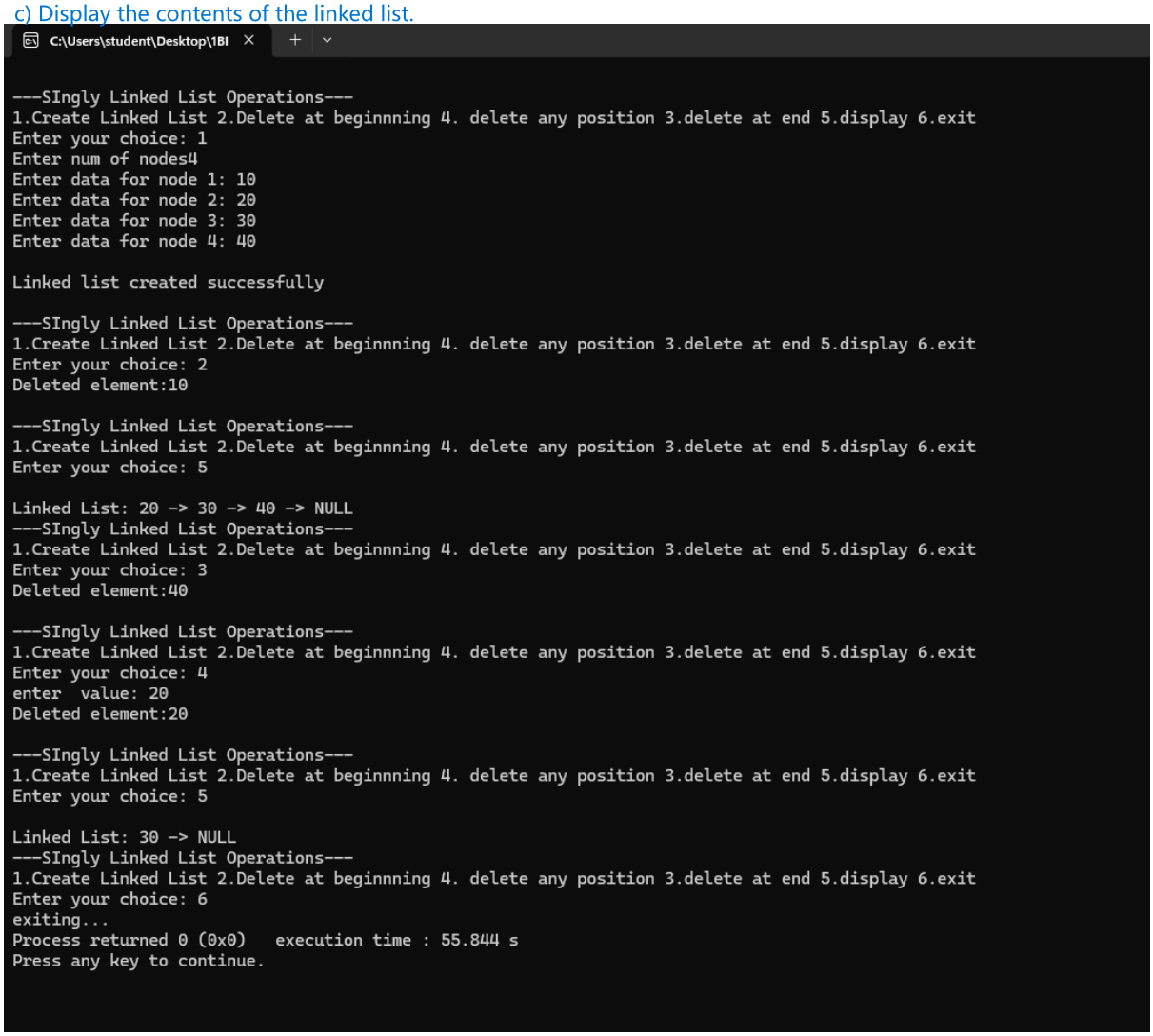
Testcase | Test Result

Accepted Runtime: 0 ms


leetcode.com/problems/middle-of-the-linked-list/submissions/1861561125/
Problem List
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 36 / 36 testcases passed
SaneyVasudhaSree006 submitted at Dec 21, 2025 18:46
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
8.51 MB | Beats 23.34%
Analyze Complexity
150%
100%
50%
0%
1ms
2ms
Code | C
/**
* Definition for singly-linked list.
*/
struct ListNode* middleNode(struct ListNode* head) {
struct ListNode *slow = head;
Code
C
Auto
*/
struct ListNode* middleNode(struct ListNode* head) {
Saved
Ln 15, Col 1
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

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.

Lab 5a



<global> deleteSpecific(int value) : void

art here | linkedlistinsertion.c | lab4b.c

```c
#include<stdio.h>
#include<stdlib.h>
struct Node{
int data;
struct Node *next;
};
struct Node *head=NULL;
void createList(int n){
struct Node *newNode,*temp;
int data,i;
if(n<=0){printf("Number of nodes should be greater than 0\n");
return ;}
for(i=1;i<=n;i++)
{
    newNode=(struct Node *)malloc(sizeof(struct Node));
    if(newNode==NULL){
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter data for node %d: ",i);
    scanf("%d",&data);
    newNode ->data=data;
    newNode->next=NULL;
    if(head==NULL){
        head=newNode;
    }
    else{
        temp->next=newNode;
    }
    temp=newNode;
    }
    printf("\nLinked list created successfully\n");
}
void deleteFirst(){
struct Node *temp;
if(head==NULL){
```

Start here | linkedlistinsertion.c | lab4b.c

```c
    head=head->next;
    printf("Deleted element:%d\n",temp->data);
    free(temp);
    return;
}
while(temp!=NULL && temp->data!=value){
    prev=temp;
    temp=temp->next;
    printf("Deleted element:%d\n,temp->data");
    free(temp);
}
}
void displayList(){
struct Node *temp=head;
if(head==NULL){
    printf("List is empty\n");
}
printf("\nLinked List: ");
while (temp!=NULL){
    printf("%d -> ",temp->data);
    temp=temp->next;
}
printf("NULL");
}
int main(){
int choice,n,value;
while(1){
    printf("\n---SIngly Linked List Operations---\n");
    printf("1.Create Linked List 2.Delete at beginnning 4. delete any position 3.delete at end 5.display 6.exit\n");

    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
    case 1:printf("Enter num of nodes");
           scanf("%d" ,&n);
           createList(n);
```

linkedlistinsertion.c | lab4b.c

```c
void deleteFirst(){
struct Node *temp;
if(head==NULL){
    printf("List is empty.Nothing to delete.\n");
    return;
}
temp=head;
head=head->next;
printf("Deleted element:%d\n",temp->data);
free(temp);
}
void deleteLast(){
struct Node *temp,*prev;
if(head==NULL){
    printf("List is empty .Nothing to delete\n");
    return;
}
temp=head;
while(temp->next!=NULL){
    prev=temp;
    temp=temp->next;
}
printf("Deleted element:%d\n",temp->data);
prev->next=NULL;
free(temp);
}
void deleteSpecific(int value){
struct Node *temp=head,*prev=NULL;
if(head==NULL){
    printf("List is empty .Nothing to delete\n");
    return;
}
if(head->data==value){
    head=head->next;
    printf("Deleted element:%d\n",temp->data);
    free(temp);
```

Start here | linkedlistinsertion.c | lab4b.c

```
    head=head->next;
    printf("Deleted element:%d\n",temp->data);
    free(temp);
    return;
}
while(temp!=NULL && temp->data!=value){
    prev=temp;
    temp=temp->next;
    printf("Deleted element:%d\n,temp->data");
    free(temp);
}
}
void displayList(){
struct Node *temp=head;
if(head==NULL){
    printf("List is empty\n");
}
printf("\nLinked List: ");
while (temp!=NULL){
    printf("%d -> ",temp->data);
    temp=temp->next;
}
printf("NULL");
}
int main(){
int choice,n,value;
while(1){
    printf("\n---SIngly Linked List Operations---\n");
    printf("1.Create Linked List 2.Delete at beginnning 4. delete any position 3.delete at end 5.display 6.exit\n");

    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
    case 1:printf("Enter num of nodes");
           scanf("%d" ,&n);
           createList(n);
```

ere | linkedlistinsertion.c | lab4b.c

```
int choice,n,value;
while(1){
    printf("\n---SIngly Linked List Operations---\n");
    printf("1.Create Linked List 2.Delete at beginnning 4. delete any position 3.delete at end 5.disp

    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
    case 1:printf("Enter num of nodes");
           scanf("%d" ,&n);
           createList(n);
           break;
    case 2:
            deleteFirst();
            break;
    case 4:printf("enter  value: ");
             scanf("%d",&value);
            deleteSpecific(value);
             break;
    case 3:

       deleteLast();break;
    case 5:
        displayList();
        break;
    case 6:printf("exiting...");
           exit(0);

    default:printf("Invalid choice Try again\n");
    }
}
return 0;
}
```

lab 5b leetcode

Problem List

Submit

Premium

Description | Editorial | Solutions | Submissions

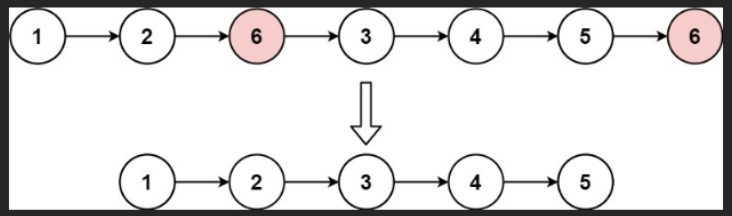
203. Remove Linked List Elements

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
```

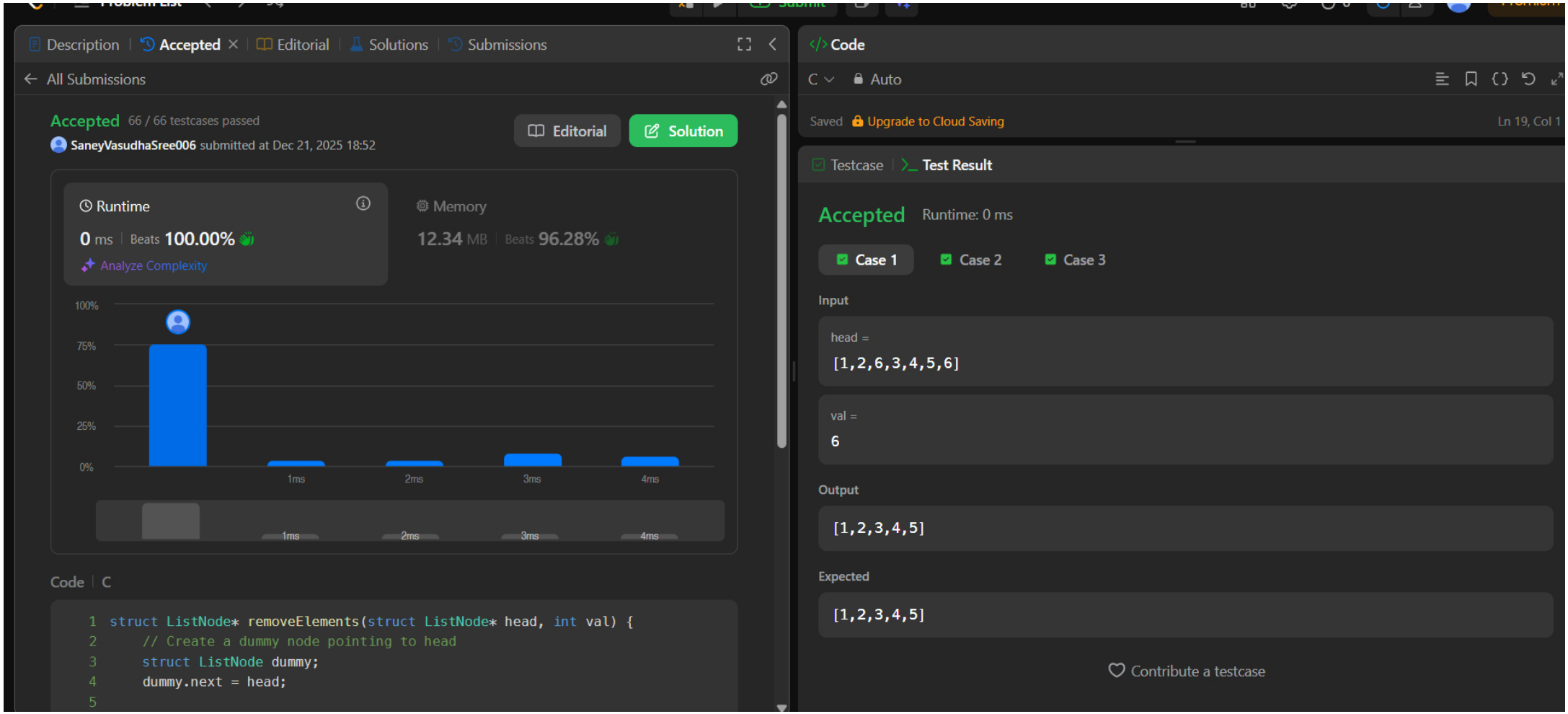
Code

C | Auto

```c
struct ListNode* removeElements(struct ListNode* head, int val) {
    // Create a dummy node pointing to head
    struct ListNode dummy;
    dummy.next = head;

    struct ListNode* current = &dummy;

    while (current->next != NULL) {
        if (current->next->val == val) {
            // Skip the node with matching value
            current->next = current->next->next;
        } else {
            current = current->next;
        }
    }

    return dummy.next;
}

```

Saved | Upgrade to Cloud Saving

Ln 19, Col 1


Description | Accepted | Editorial | Solutions | Submissions
All Submissions
Accepted 66 / 66 testcases passed
SaneyVasudhaSree006 submitted at Dec 21, 2025 18:52
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
12.34 MB | Beats 96.28%
100%
75%
50%
25%
0%
1ms
2ms
3ms
4ms
Code | C
1 struct ListNode* removeElements(struct ListNode* head, int val) {
2 // Create a dummy node pointing to head
3 struct ListNode dummy;
4 dummy.next = head;
5
Code
C
Auto
Saved
Upgrade to Cloud Saving
Ln 19, 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]
Contribute a testcase

lab 6a

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

createList(int n) : Node

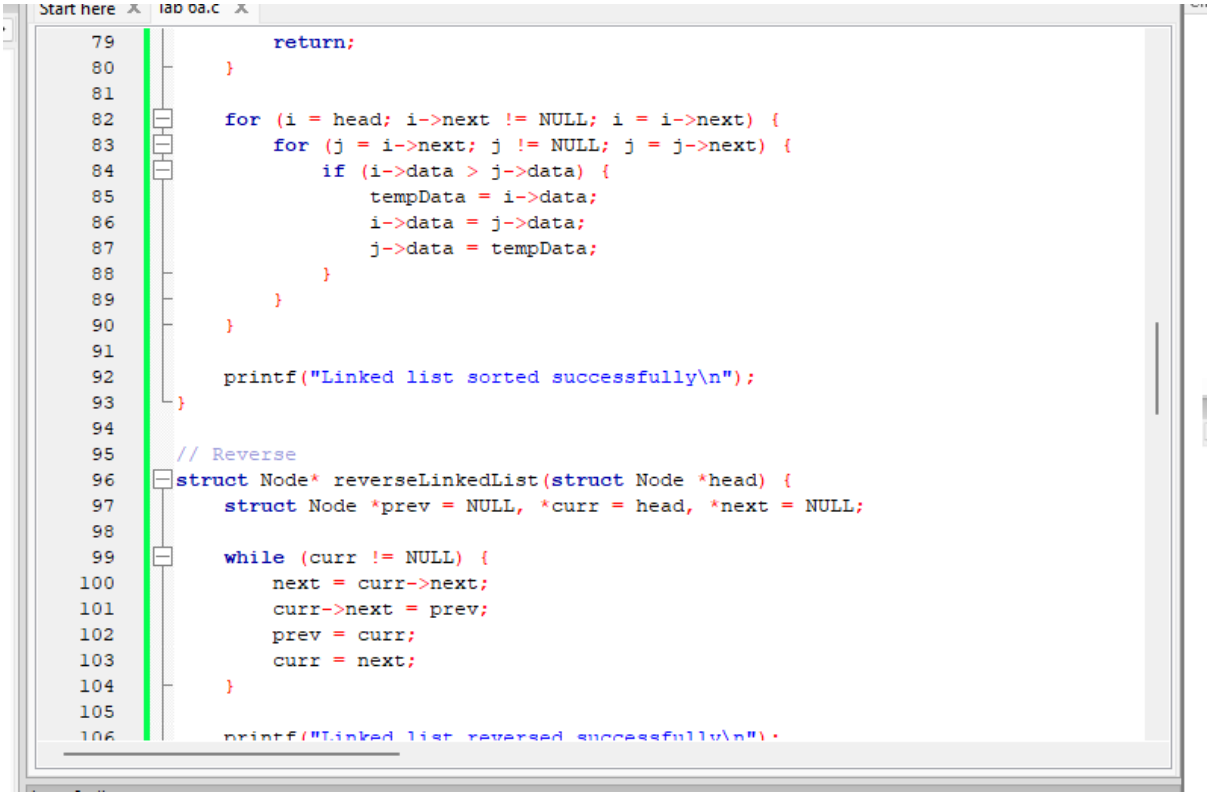
Start here | lab 6a.c

```
28      }
29
30      for (int i = 1; i <= n; i++) {
31          newNode = (struct Node*)malloc(sizeof(struct Node));
32          if (newNode == NULL) {
33              printf("Memory allocation failed\n");
34              return head;
35          }
36
37          printf("Enter data for node %d: ", i);
38          scanf("%d", &data);
39
40          newNode->data = data;
41          newNode->next = NULL;
42
43          if (head == NULL)
44              head = newNode;
45          else
46              temp->next = newNode;
47
48          temp = newNode;
49      }
50
51      printf("Linked list created successfully\n");
52      return head;
53  }
54
55  // Display
```

Start here | lab 6a.c

```
52      return head;
53  }
54
55  // Display
56  void displayList(struct Node *head) {
57      struct Node *temp = head;
58
59      if (head == NULL) {
60          printf("List is empty\n");
61          return;
62      }
63
64      printf("Linked List: ");
65      while (temp != NULL) {
66          printf("%d -> ", temp->data);
67          temp = temp->next;
68      }
69      printf("NULL\n");
70  }
71
72  // Sort
73  void sortLinkedList(struct Node *head) {
74      struct Node *i, *j;
75      int tempData;
76
77      if (head == NULL) {
78          printf("List is empty, cannot sort.\n");
79          return;
```

Logs & others


Start here
lab 8a.c
79 return;
80 }
81
82 for (i = head; i->next != NULL; i = i->next) {
83 for (j = i->next; j != NULL; j = j->next) {
84 if (i->data > j->data) {
85 tempData = i->data;
86 i->data = j->data;
87 j->data = tempData;
88 }
89 }
90 }
91
92 printf("Linked list sorted successfully\n");
93 }
94
95 // Reverse
96 struct Node* reverseLinkedList(struct Node *head) {
97 struct Node *prev = NULL, *curr = head, *next = NULL;
98
99 while (curr != NULL) {
100 next = curr->next;
101 curr->next = prev;
102 prev = curr;
103 curr = next;
104 }
105
106 printf("Linked list reversed successfully\n");

```
106     printf("Linked list reversed successfully\n");
107     return prev;
108 }
109
110 // Concatenate
111 struct Node* concatenateLinkedList(struct Node *head1, struct Node *head2) {
112     struct Node *temp;
113
114     if (head1 == NULL)
115         return head2;
116
117     temp = head1;
118     while (temp->next != NULL)
119         temp = temp->next;
120
121     temp->next = head2;
122
123     printf("Linked lists concatenated successfully\n");
124     return head1;
125 }
126
127 // Menu
128 int main() {
129     int choice, n;
130
131     while (1) {
132         printf("\n================================\n");
133         printf("          LINKED LIST MENU      \n");
```

createList(int n) : Node

Start here | lab 6a.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Node {
5      int data;
6      struct Node *next;
7  };
8
9  // Global heads
10 struct Node *head1 = NULL;
11 struct Node *head2 = NULL;
12
13 // Function prototypes
14 struct Node* createList(int n);
15 void displayList(struct Node *head);
16 void sortLinkedList(struct Node *head);
17 struct Node* reverseLinkedList(struct Node *head);
18 struct Node* concatenateLinkedList(struct Node *head1, struct Node *head2);
19
20 // Create list
21 struct Node* createList(int n) {
22     struct Node *head = NULL, *newNode, *temp;
23     int data;
24
25     if (n <= 0) {
26         printf("Number of nodes should be greater than 0\n");
27         return NULL;
28     }
```

```
133         printf("        LINKED LIST MENU       \n");
134         printf("==============================\n");
135         printf("1. Create List 1\n");
136         printf("2. Create List 2\n");
137         printf("3. Display List 1\n");
138         printf("4. Display List 2\n");
139         printf("5. Sort List 1\n");
140         printf("6. Reverse List 1\n");
141         printf("7. Concatenate List 1 + List 2\n");
142         printf("8. Exit\n");
143         printf("Enter choice: ");
144         scanf("%d", &choice);
145
146         switch (choice) {
147         case 1:
148             printf("Enter number of nodes for List 1: ");
149             scanf("%d", &n);
150             head1 = createList(n);
151             break;
152
153         case 2:
154             printf("Enter number of nodes for List 2: ");
155             scanf("%d", &n);
156             head2 = createList(n);
157             break;
158
159         case 3:
160             displayList(head1);
```

Start here | lab 6a.c

```
160             displayList(head1);
161             break;
162
163         case 4:
164             displayList(head2);
165             break;
166
167         case 5:
168             sortLinkedList(head1);
169             break;
170
171         case 6:
172             head1 = reverseLinkedList(head1);
173             break;
174
175         case 7:
176             head1 = concatenateLinkedList(head1, head2);
177             printf("After concatenation:\n");
178             displayList(head1);
179             break;
180
181         case 8:
182             printf("Exiting program...\n");
183             exit(0);
184
185         default:
186             printf("Invalid choice. Try again.\n");
187         }
```

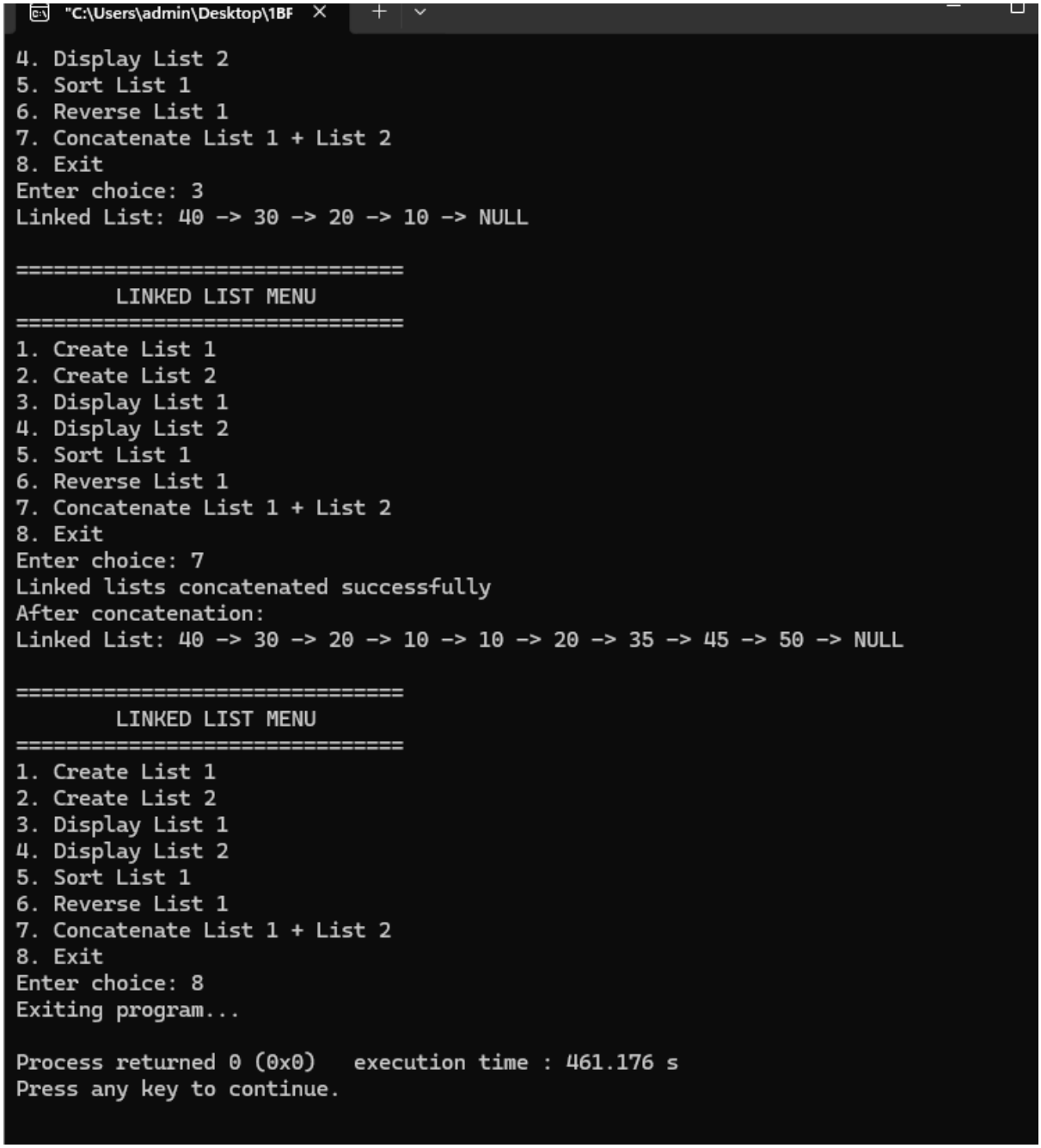
Logs & others

```
            break;

        case 6:
            head1 = reverseLinkedList(head1);
            break;

        case 7:
            head1 = concatenateLinkedList(head1, head2);
            printf("After concatenation:\n");
            displayList(head1);
            break;

        case 8:
            printf("Exiting program...\n");
            exit(0);

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

    return 0;
}
```

```
==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 1
Enter number of nodes for List 1: 4
Enter data for node 1: 40
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 10
Linked list created successfully

==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 2
Enter number of nodes for List 2: 5
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 35
Enter data for node 4: 45
Enter data for node 5: 50
Linked list created successfully

==============================
```

```
==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 3
Linked List: 40 -> 20 -> 30 -> 10 -> NULL

==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 4
Linked List: 10 -> 20 -> 35 -> 45 -> 50 -> NULL

==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 5
```

```
8. Exit
Enter choice: 5
Linked list sorted successfully

==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 3
Linked List: 10 -> 20 -> 30 -> 40 -> NULL

==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 6
Linked list reversed successfully

==============================
        LINKED LIST MENU
==============================
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
```



Lab 6b

lab 6b

#include <stdio.h>

#include <stdlib.h>

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

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

```
// Stack pointers
struct Node *top = NULL;

// Queue pointers
struct Node *front = NULL;
struct Node *rear = NULL;



// Function to create a new node
struct Node* createNode(int value) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed!\n");
        exit(0);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

////////////////////////////////////////////////////////////
//                STACK OPERATIONS                //
////////////////////////////////////////////////////////////

// PUSH
void push(int value) {
    struct Node *newNode = createNode(value);
    newNode->next = top;
    top = newNode;
    printf("Pushed: %d\n", value);
```

```
}

// POP
void pop() {
   if (top == NULL) {
      printf("Stack Underflow!\n");
      return;
   }
   struct Node *temp = top;
   printf("%d popped from the stack\n", temp->data);
   top = top->next;
   free(temp);
}

// DISPLAY STACK
void displayStack() {
   struct Node *temp = top;
   if (temp== NULL) {
      printf("Stack is Empty!\n");
      return;
   }

   printf("Stack (top to bottom) elements are: ");
   while (temp!= NULL) {
      printf("%d ", temp->data);
      temp=temp->next;
   }
   printf("\n");
}

////////////////////////////////////////////////////////////
```

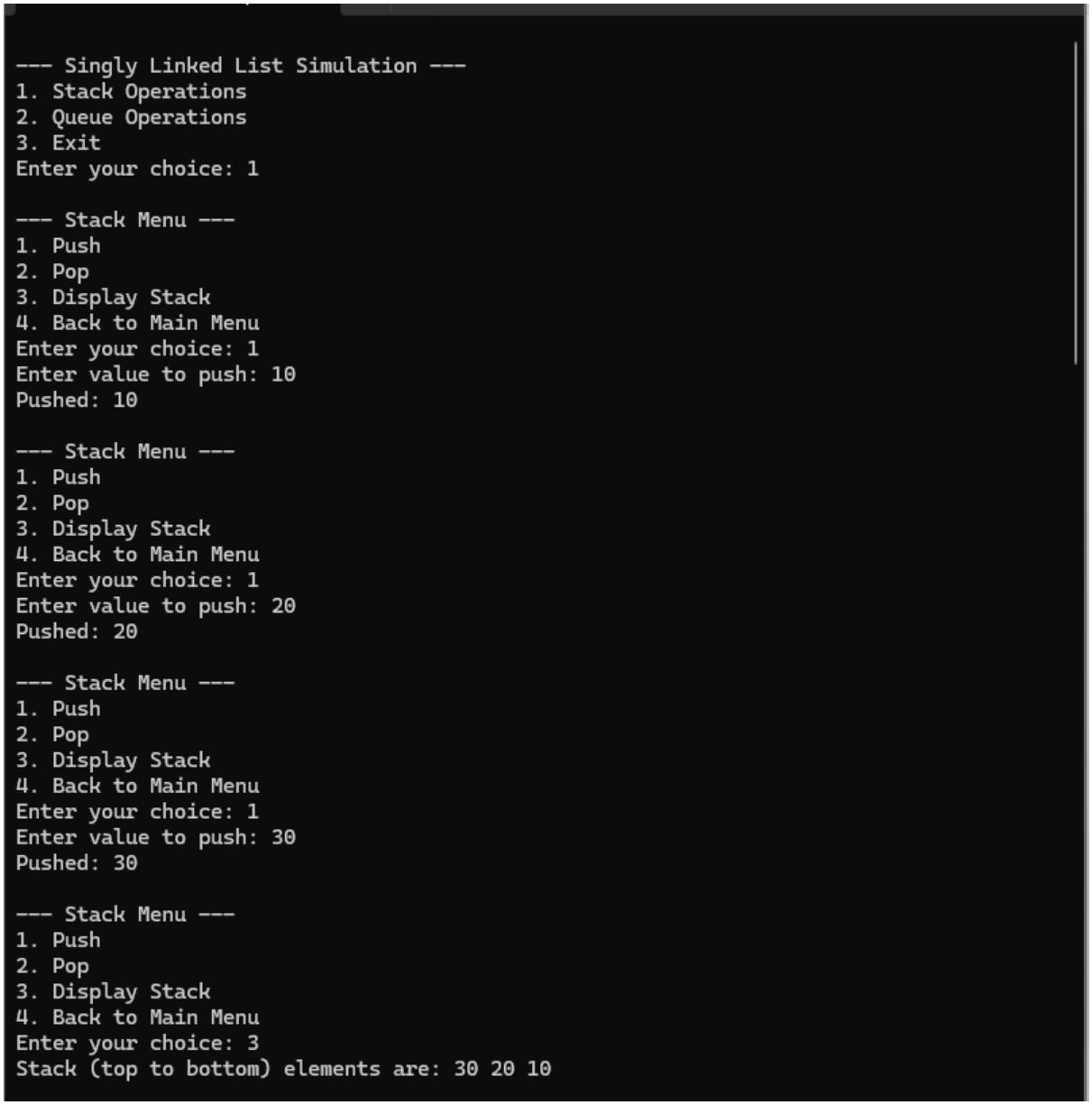
```
//                QUEUE OPERATIONS                //
////////////////////////////////////////////////////////

// ENQUEUE
void enqueue(int value) {
   struct Node *newNode = createNode(value);

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

   printf("%d enqueued to the queue\n", value);
}

// DEQUEUE
void dequeue() {
   if (front == NULL) {
      printf("Queue is empty\n");
      return;
   }

   struct Node *temp = front;
   printf("%d dequeued from queue\n", temp->data);

   front = front->next;
   if (front == NULL)
      rear = NULL;
```

```
    free(temp);
}

// DISPLAY QUEUE
void displayQueue() {
    struct Node *temp = front;
    if (temp == NULL) {
        printf("Queue is Empty!\n");
        return;
    }

    printf("Queue (front to rear) elements are: ");
    while (temp!= NULL) {
        printf("%d ", temp->data);
        temp=temp->next;
    }
    printf("\n");
}

////////////////////////////////////////////////////////////

int main() {
    int choice, value, ch;

    while (1) {
        printf("\n--- Singly Linked List Simulation ---\n");
        printf("1. Stack Operations\n");
        printf("2. Queue Operations\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {

   // ---------------- STACK MENU ----------------
   case 1:
      while (1) {
         printf("\n--- Stack Menu ---\n");
         printf("1. Push\n");
         printf("2. Pop\n");
         printf("3. Display Stack\n");
         printf("4. Back to Main Menu\n");
         printf("Enter your choice: ");
         scanf("%d", &ch);

         switch (ch) {
            case 1:
               printf("Enter value to push: ");
               scanf("%d", &value);
               push(value);
               break;

            case 2:
               pop();
               break;

            case 3:
               displayStack();
               break;

            case 4:
               goto main_menu;
```

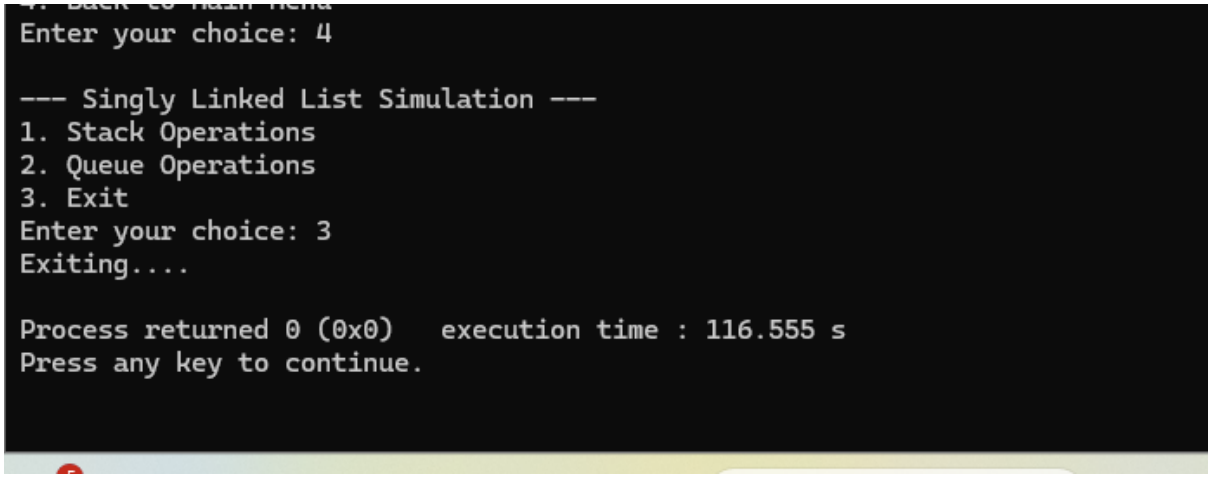
```
            default:
                printf("Invalid Choice!\n");
        }
    }
    break;

// ---------- QUEUE MENU ----------
case 2:
    while (1) {
        printf("\n--- Queue Menu ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display Queue\n");
        printf("4. Back to Main Menu\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;

            case 2:
                dequeue();
                break;

            case 3:
                displayQueue();
                break;
```

```
                case 4:
                    goto main_menu;
                default:
                    printf("Invalid Choice!\n");
                }
            }
            break;

        // ---------- EXIT ----------
        case 3:
            printf("Exiting....\n");
            exit(0);

        default:
            printf("Invalid choice!\n");
    }
main_menu: ;
}
return 0;
}
```


--- Singly Linked List Simulation ---
1. Stack Operations
2. Queue Operations
3. Exit
Enter your choice: 1
--- Stack Menu ---
1. Push
2. Pop
3. Display Stack
4. Back to Main Menu
Enter your choice: 1
Enter value to push: 10
Pushed: 10
--- Stack Menu ---
1. Push
2. Pop
3. Display Stack
4. Back to Main Menu
Enter your choice: 1
Enter value to push: 20
Pushed: 20
--- Stack Menu ---
1. Push
2. Pop
3. Display Stack
4. Back to Main Menu
Enter your choice: 1
Enter value to push: 30
Pushed: 30
--- Stack Menu ---
1. Push
2. Pop
3. Display Stack
4. Back to Main Menu
Enter your choice: 3
Stack (top to bottom) elements are: 30 20 10

```
--- Stack Menu ---
1. Push
2. Pop
3. Display Stack
4. Back to Main Menu
Enter your choice: 2
30 popped from the stack

--- Stack Menu ---
1. Push
2. Pop
3. Display Stack
4. Back to Main Menu
Enter your choice: 4

--- Singly Linked List Simulation ---
1. Stack Operations
2. Queue Operations
3. Exit
Enter your choice: 4
Invalid choice!

--- Singly Linked List Simulation ---
1. Stack Operations
2. Queue Operations
3. Exit
Enter your choice: 2

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
4. Back to Main Menu
Enter your choice: 1
Enter value to enqueue: 5
5 enqueued to the queue

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
```

```
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
4. Back to Main Menu
Enter your choice: 1
Enter value to enqueue: 4
4 enqueued to the queue

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
4. Back to Main Menu
Enter your choice: 1
Enter value to enqueue: 6
6 enqueued to the queue

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
4. Back to Main Menu
Enter your choice: 3
Queue (front to rear) elements are: 5 4 6

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
4. Back to Main Menu
Enter your choice: 2
5 dequeued from queue

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display Queue
4. Back to Main Menu
Enter your choice: 4
```


Enter your choice: 4
--- Singly Linked List Simulation ---
1. Stack Operations
2. Queue Operations
3. Exit
Enter your choice: 3
Exiting....
Process returned 0 (0x0)   execution time : 116.555 s
Press any key to continue.

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, *tail = NULL;

void createList(int n) {
    int i, data;
    struct Node *newNode;
    for (i = 1; i <= n; i++) {
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->prev = newNode->next = NULL;
        if (head == NULL) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
}

void insertAtFront(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = head;
    if (head == NULL)
        head = tail = newNode;
    else {
        head->prev = newNode;
        head = newNode;
    }
    printf("Inserted %d at front\n",data);
}

void insertAtEnd(int data) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = tail;
    if (tail == NULL)
        head = tail = newNode;
    else {
        tail->next = newNode;
        tail = newNode;
    }
    printf("Inserted %d at end\n",data);
}

void insertAtPosition(int data, int pos) {
    int i;
    struct Node *newNode, *temp = head;
    if (pos == 1) {
        insertAtFront(data);
```

```
        return;
    }
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    for (i = 1; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;
    if (temp == NULL || temp->next == NULL) {
        insertAtEnd(data);
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    temp->next->prev = newNode;
    temp->next = newNode;
    printf("Inserted %d at position %d\n",data,pos);
}

void deleteAtFront() {
    struct Node *temp;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    temp = head;
    head = head->next;
    if (head != NULL)
        head->prev = NULL;
    else
        tail = NULL;
        printf("Deleted %d at front\n",temp->data);
    free(temp);
```

```
}

void deleteAtEnd() {
   struct Node *temp;
   if (tail == NULL) {
      printf("List is empty!\n");
      return;
   }
   temp = tail;
   tail = tail->prev;
   if (tail != NULL)
      tail->next = NULL;
   else
      head = NULL;
      printf("Deleted %d at end\n",temp->data);
   free(temp);
}

void deleteByValue(int value) {
   struct Node *temp = head;
   if (head == NULL) {
      printf("List is empty!\n");
      return;
   }
   while (temp != NULL && temp->data != value)
      temp = temp->next;
   if (temp == NULL) {
      printf("Value not found!\n");
      return;
   }
   if (temp == head)
```

```
        deleteAtFront();
    else if (temp == tail)
        deleteAtEnd();
    else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        printf("Deleted %d \n",temp->data);
        free(temp);
    }
}

void search(int value) {
    struct Node *temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Value %d found at position %d\n", value, pos);
            return;
        }
        temp = temp->next;
        pos++;
    }
    printf("Value %d not found in the list.\n", value);
}

void displayForward() {
    struct Node *temp = head;
    printf("List (Forward): ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
```

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

void displayBackward() {
    struct Node *temp = tail;
    printf("List (Backward): ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->prev;
    }
    printf("NULL\n");
}

int main() {
    int choice, n, data, pos, value;

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

    while (1) {
        printf("\n--- DOUBLY LINKED LIST MENU ---\n");
        printf("1. Display Forward\n");
        printf("2. Display Backward\n");
        printf("3. Insert at Front\n");
        printf("4. Insert at End\n");
        printf("5. Insert at Position\n");
        printf("6. Delete at Front\n");
        printf("7. Delete at End\n");
        printf("8. Delete by Value\n");
```

```
printf("9. Search\n");
printf("10. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {

   case 1:
      displayForward();
      break;

   case 2:
      displayBackward();
      break;

   case 3:
      printf("Enter data to insert at front: ");
      scanf("%d", &data);
      insertAtFront(data);
      break;

   case 4:
      printf("Enter data to insert at end: ");
      scanf("%d", &data);
      insertAtEnd(data);
      break;

   case 5:
      printf("Enter data: ");
      scanf("%d", &data);
      printf("Enter position: ");
```

```
            scanf("%d", &pos);
            insertAtPosition(data, pos);
            break;

        case 6:
            deleteAtFront();
            break;

        case 7:
            deleteAtEnd();
            break;

        case 8:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteByValue(value);
            break;

        case 9:
            printf("Enter value to search: ");
            scanf("%d", &value);
            search(value);
            break;

        case 10:
            printf("Exiting program...\n");
            exit(0);

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

```
    }

    return 0;
}
```

```
Enter number of nodes to create: 4
Enter data for node 1: 40
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 20

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 1
List (Forward): 40 <-> 20 <-> 30 <-> 20 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 2
List (Backward): 20 <-> 30 <-> 20 <-> 40 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
```

```
8. Delete by Value
9. Search
10. Exit
Enter your choice: 3
Enter data to insert at front: 5
Inserted 5 at front

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 4
Enter data to insert at end: 50
Inserted 50 at end

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 5
Enter data: 4
Enter position: 2
Inserted 4 at position 2

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
```

```
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 1
List (Forward): 5 <-> 4 <-> 40 <-> 20 <-> 30 <-> 20 <-> 50 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 6
Deleted 5 at front

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 7
Deleted 50 at end
```

```
Deleted 50 at end

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 8
Enter value to delete: 20
Deleted 20

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 1
List (Forward): 4 <-> 40 <-> 30 <-> 20 <-> NULL

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
```

```
--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 9
Enter value to search: 40
Value 40 found at position 2

--- DOUBLY LINKED LIST MENU ---
1. Display Forward
2. Display Backward
3. Insert at Front
4. Insert at End
5. Insert at Position
6. Delete at Front
7. Delete at End
8. Delete by Value
9. Search
10. Exit
Enter your choice: 10
Exiting program...

Process returned 0 (0x0)   execution time : 106.506 s
Press any key to continue.
```

Lab 7b leetcode

leetcode.com/problems/linked-list-cycle/

Problem List | Submit | Premium

Description | Editorial | Solutions | Submissions

141. Linked List Cycle

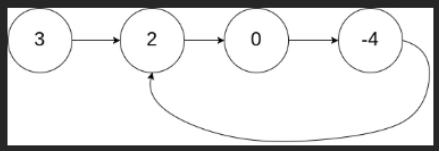
Easy | Topics | Companies

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` *if there is a cycle in the linked list.* Otherwise, return `false`.

Example 1:




```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).
```

Example 2:

17.2K | 474 | 162 Online

Code

C | Auto

```c

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;
}

```

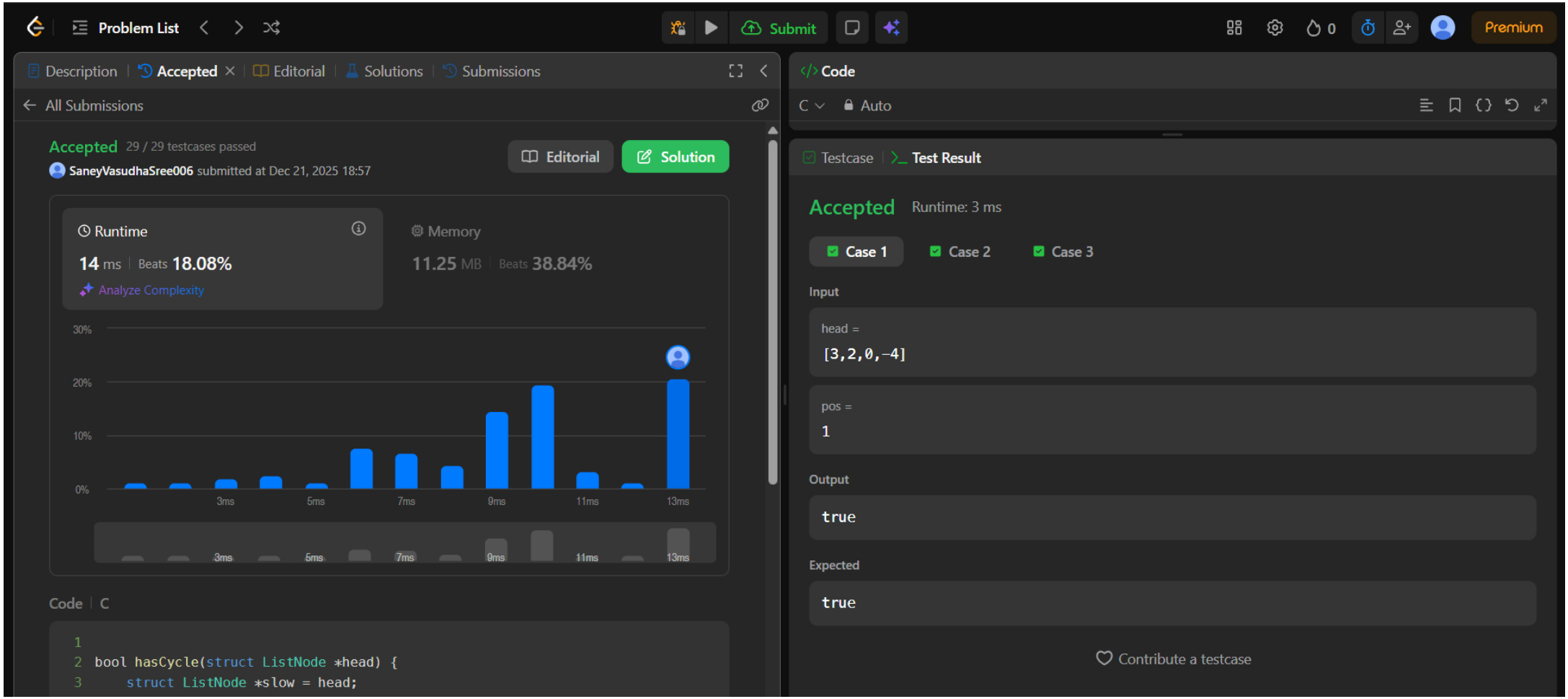
Saved | Ln 17, Col 1

Testcase | Test Result

Accepted Runtime: 3 ms

Case 1 | Case 2 | Case 3

Input


Problem List
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 29 / 29 testcases passed
SaneyVasudhaSree006 submitted at Dec 21, 2025 18:57
Editorial
Solution
Runtime
14 ms | Beats 18.08%
Analyze Complexity
Memory
11.25 MB | Beats 38.84%
30%
20%
10%
0%
3ms
5ms
7ms
9ms
11ms
13ms
Code | C
1
2 bool hasCycle(struct ListNode *head) {
3 struct ListNode *slow = head;
Code
C
Auto
Testcase | Test Result
Accepted Runtime: 3 ms
Case 1
Case 2
Case 3
Input
head =
[3,2,0,-4]
pos =
1
Output
true
Expected
true
Contribute a testcase

Lab 8a

Write a program
a) ToconstructabinarySearchtree.
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 if (value > root->data)
        root->right = insert(root->right, value);

    return root;
}

/* Inorder Traversal: Left -> Root -> Right */
void inorder(struct Node *root) {
    if (root == NULL)
        return;
```

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

/* Preorder Traversal: Root -> Left -> Right */
void preorder(struct Node *root) {
    if (root == NULL)
        return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

/* Postorder Traversal: Left -> Right -> Root */
void postorder(struct Node *root) {
    if (root == NULL)
        return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

void display(struct Node *root) {
    printf("BST Elements (Inorder): ");
    inorder(root);
    printf("\n");
}

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

```
int choice, value;

while (1) {
    printf("\n--- Binary Search Tree Menu ---\n");
    printf("1. Insert into BST\n");
    printf("2. In-order Traversal\n");
    printf("3. Pre-order Traversal\n");
    printf("4. Post-order Traversal\n");
    printf("5. Display BST\n");
    printf("6. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            root = insert(root, value);
            break;

        case 2:
            printf("In-order Traversal: ");
            inorder(root);
            printf("\n");
            break;

        case 3:
            printf("Pre-order Traversal: ");
            preorder(root);
            printf("\n");
            break;
```

```
            case 4:
                printf("Post-order Traversal: ");
                postorder(root);
                printf("\n");
                break;

            case 5:
                display(root);
                break;

            case 6:
                printf("Exiting....");
                exit(0);

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

    return 0;
}
```


C:\Users\admin\Desktop\IBF
--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 40
--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 20
--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 1
Enter value to insert: 30
--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 2
In-order Traversal: 20 30 40

```
In-order Traversal: 20 30 40

--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 3
Pre-order Traversal: 40 20 30

--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 4
Post-order Traversal: 30 20 40

--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 5
BST Elements (Inorder): 20 30 40

--- Binary Search Tree Menu ---
1. Insert into BST
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display BST
6. Exit
Enter choice: 6
Exiting....
```

Lab 8b leetcode

Description | Editorial | Solutions | Submissions

617. Merge Two Binary Trees

Easy | Topics | Companies

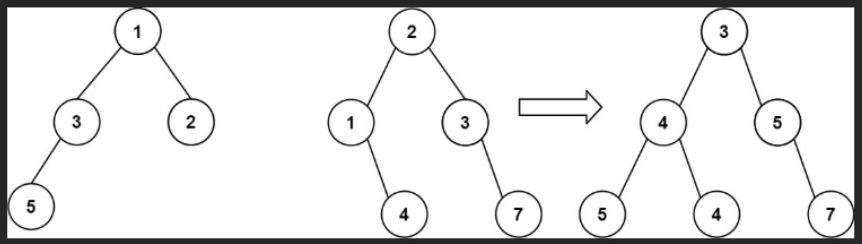
You are given two binary trees `root1` and `root2`.

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]

9K | 41 | 15 Online

Code

C | Auto

```c
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
    if (root1 == NULL)
        return root2;

    if (root2 == NULL)
        return root1;

    // Both nodes exist
    struct TreeNode* merged = malloc(sizeof(struct TreeNode));
    merged->val = root1->val + root2->val;
    merged->left = mergeTrees(root1->left, root2->left);
    merged->right = mergeTrees(root1->right, root2->right);

    return merged;
}

```

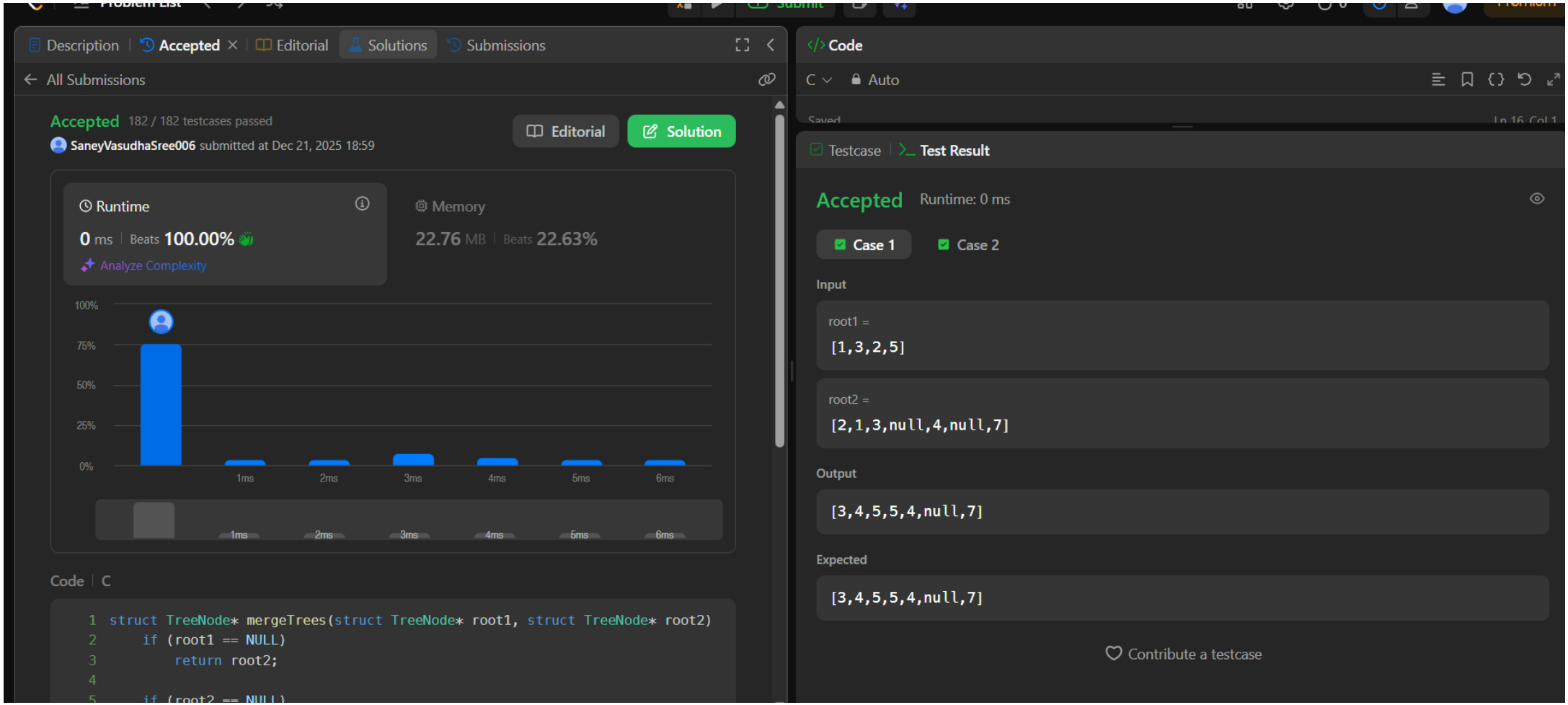
Saved | Ln 16, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

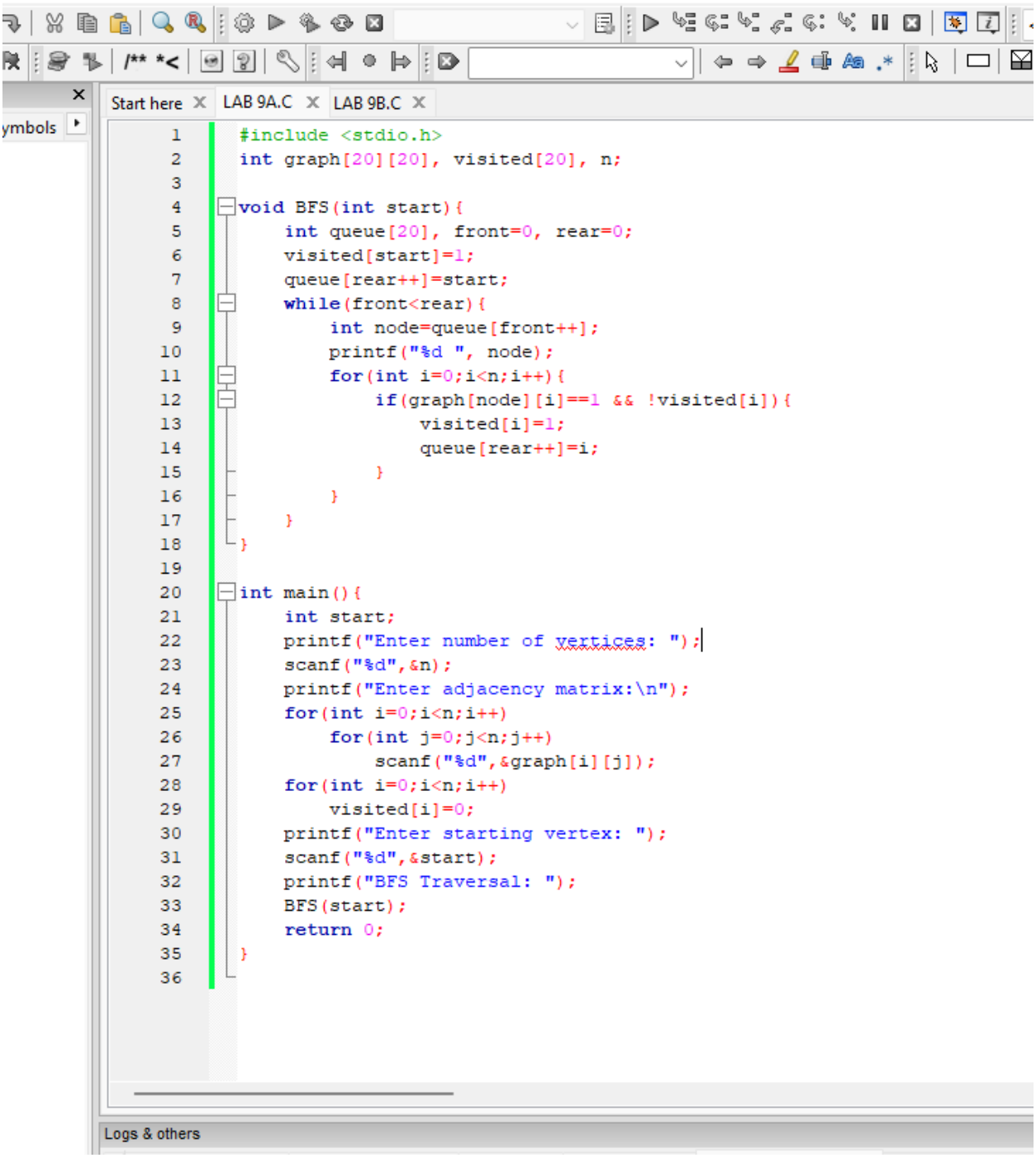
Input


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 182 / 182 testcases passed
SaneyVasudhaSree006 submitted at Dec 21, 2025 18:59
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
22.76 MB | Beats 22.63%
100%
75%
50%
25%
0%
1ms
2ms
3ms
4ms
5ms
6ms
Code | C
1 struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2)
2 if (root1 == NULL)
3 return root2;
4
Code
C
Auto
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]
Contribute a testcase

lab 9a

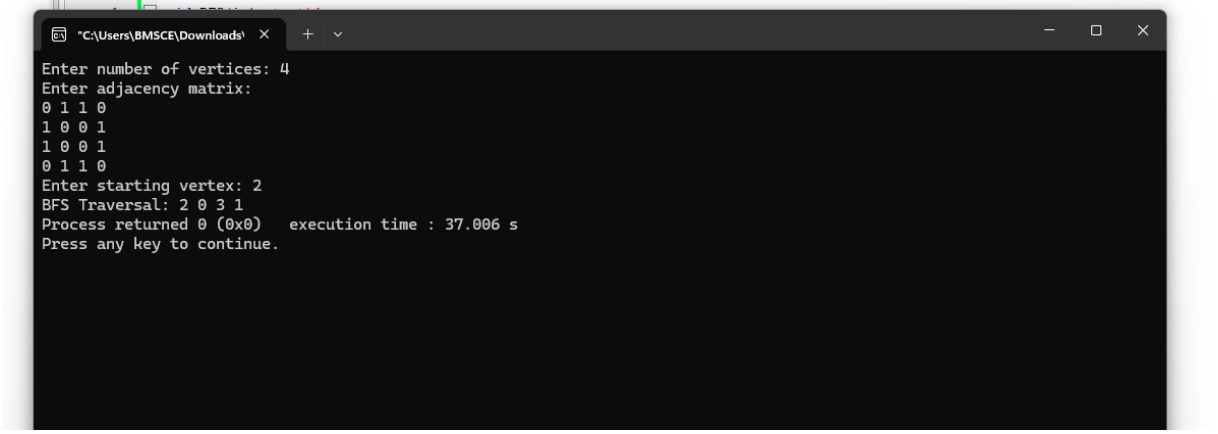
a) Write a program to traverse a graph using BFS method

Lab 9A



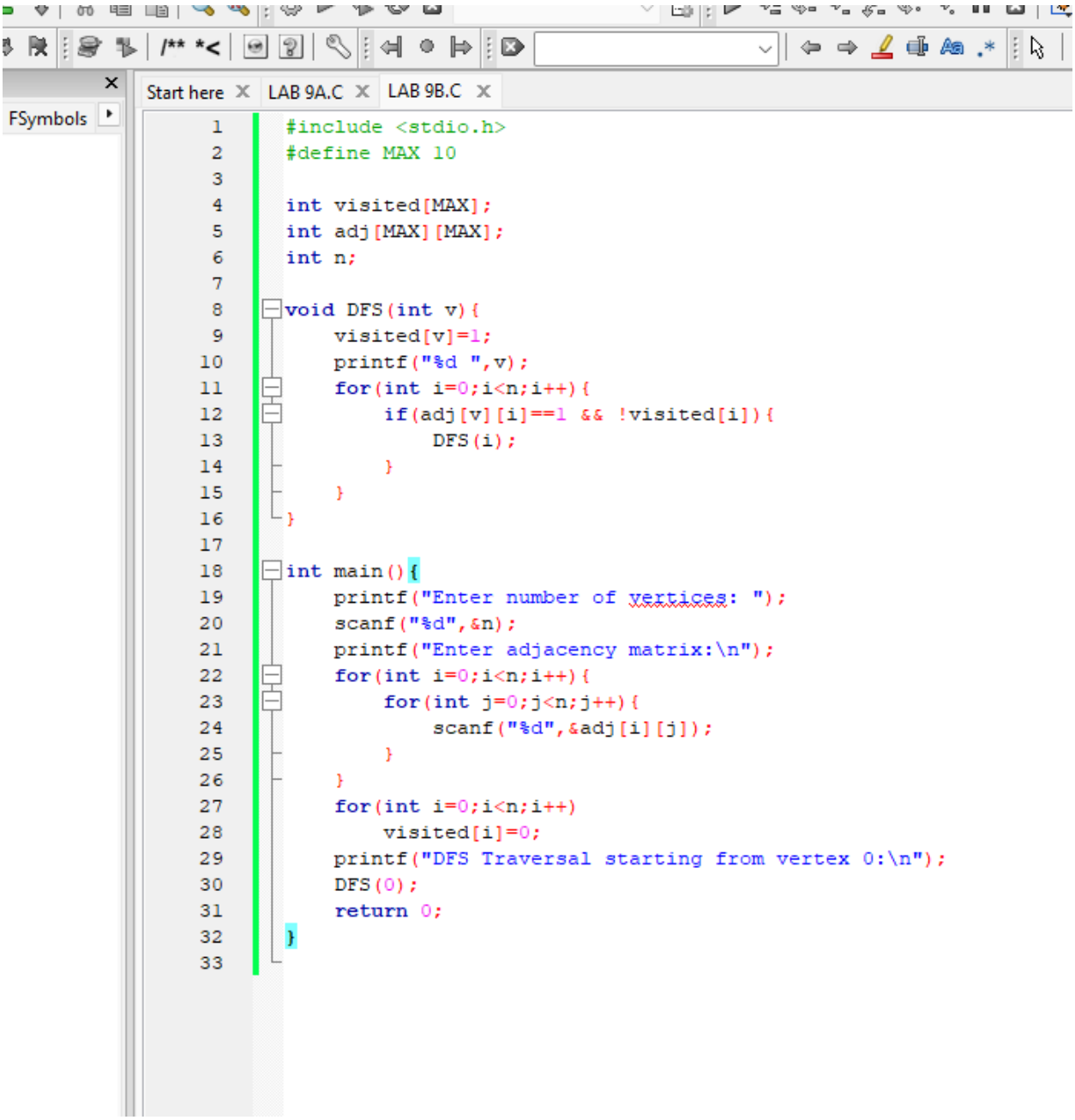

```c
#include <stdio.h>
int graph[20][20], visited[20], n;

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;
}
```

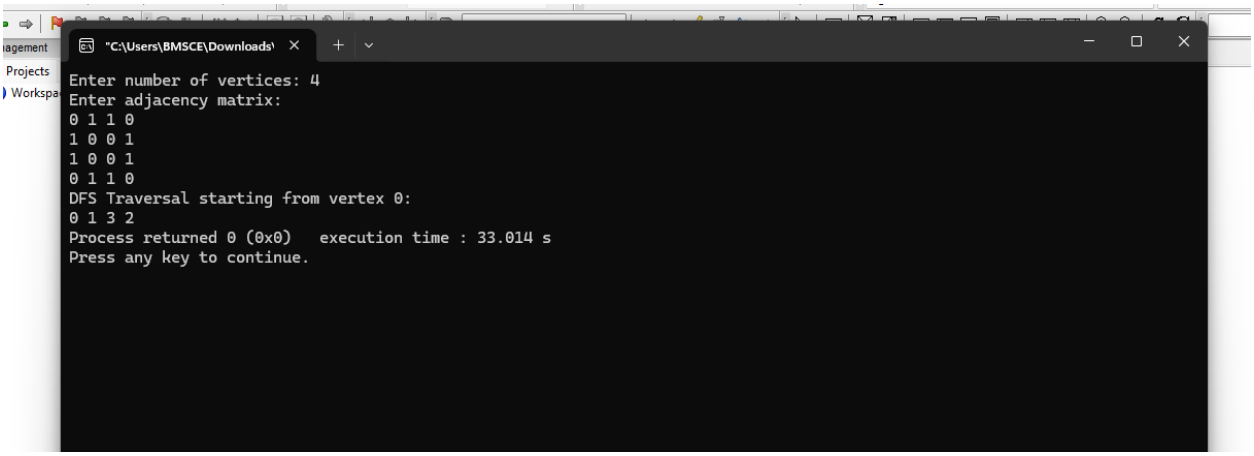
```
2   int graph[20][20], visited[20], n;
3
```



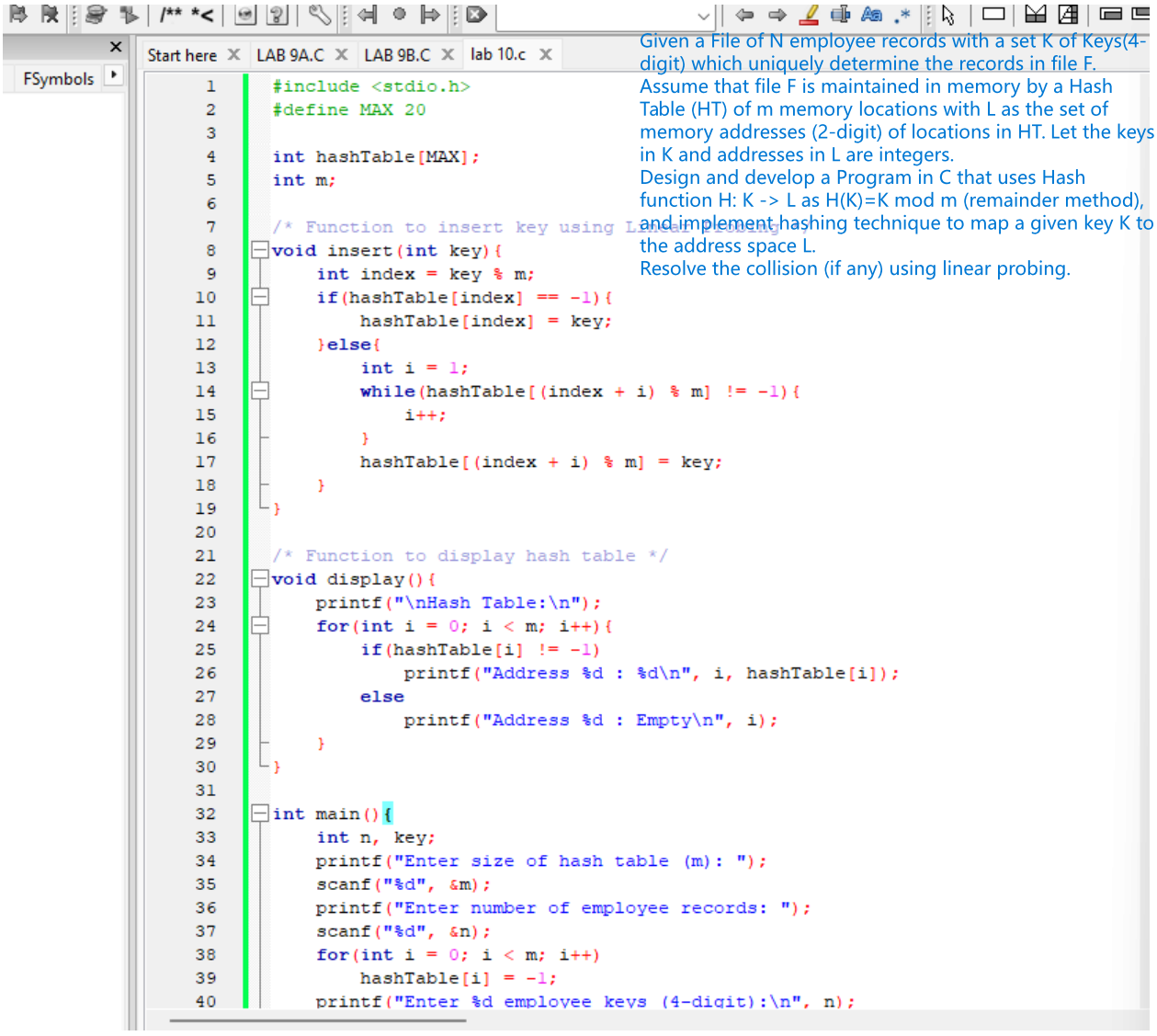
lab 9b

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




"C:\Users\BMSCE\Downloads\
Enter number of vertices: 4
Enter adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
DFS Traversal starting from vertex 0:
0 1 3 2
Process returned 0 (0x0)   execution time : 33.014 s
Press any key to continue.

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.

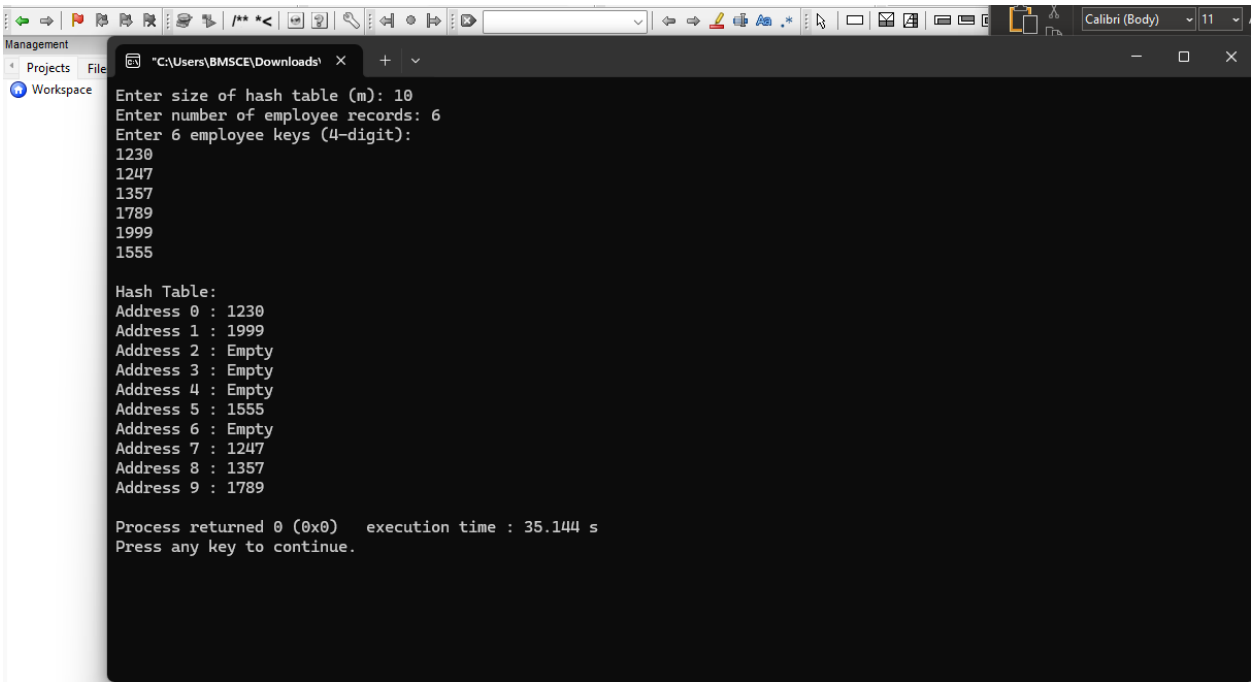


```
11          hashTable[index] = key;
12      }else{
13          int i = 1;
14          while(hashTable[(index + i) % m] != -1){
15              i++;
16          }
17          hashTable[(index + i) % m] = key;
18      }
19  }
20
21  /* Function to display hash table */
22  void display(){
23      printf("\nHash Table:\n");
24      for(int i = 0; i < m; i++){
25          if(hashTable[i] != -1)
26              printf("Address %d : %d\n", i, hashTable[i]);
27          else
28              printf("Address %d : Empty\n", i);
29      }
30  }
31
32  int main(){
33      int n, key;
34      printf("Enter size of hash table (m): ");
35      scanf("%d", &m);
36      printf("Enter number of employee records: ");
37      scanf("%d", &n);
38      for(int i = 0; i < m; i++)
39          hashTable[i] = -1;
40      printf("Enter %d employee keys (4-digit):\n", n);
41      for(int i = 0; i < n; i++){
42          scanf("%d", &key);
43          insert(key);
44      }
45      display();
46      return 0;
47  }
48
```

Logs & others

Code::Blocks | Search results | Cccc | Build log | Build messages | CppCheck/Vera++ | C

File	Line	Message
		=== Build file: "no target" in "no project" (compiler: unknown) ===


Management
Projects
File
Workspace
"C:\Users\BMSCE\Downloads\
Enter size of hash table (m): 10
Enter number of employee records: 6
Enter 6 employee keys (4-digit):
1230
1247
1357
1789
1999
1555
Hash Table:
Address 0 : 1230
Address 1 : 1999
Address 2 : Empty
Address 3 : Empty
Address 4 : Empty
Address 5 : 1555
Address 6 : Empty
Address 7 : 1247
Address 8 : 1357
Address 9 : 1789
Process returned 0 (0x0) execution time : 35.144 s
Press any key to continue.