VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

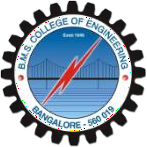
**Submitted by**

**V TANUSREE (1BM22CS313)**

**in partial fulfillment for the award of the degree of BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU) BENGALURU-560019**

**Dec 2023- March 2024**

1**|** Page

**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 **V TANUSREE(1BM22CS313)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-

24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST )**work prescribed for the said degree.

**Prof. Lakshmi Neelima Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

2**|** Page

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1 | Stack Implementation using Array | 4 |
| 2 | Conversion of Infix Expression to Postfix Expression | 8 |
| 3 | 3a. Queue Implementation using Array  3b. Circular Queue Implementation using Array | 14 |
| 4 | Singly Linked List Implementation:Insertion | 22 |
| 5 | Singly Linked List Operations: Deletion | 22 |
| 6 | 6a. Single Linked List Operations: Sorting, Reversing, Concatenating  6b. Single Linked List Simulation: Stack & Queue Operations | 37 |
| 7 | Doubly Linked List Implementation | 53 |
| 8 | Binary Search Tree Construction and Traversal | 63 |
| 9 | 9a. Graph Traversal using BFS  9b. Checking Graph Connectivity using DFS | 73 |
| 10 | Hash Table Implementation with Linear Probing | 85 |

**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. |

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 5

void push(int value); void pop();

void display();

int top=-1,stack[max]; void main(){

int choice,value; while (1) {

printf("\npick one choice:\n"); printf("1.push\n2.pop\n3.display\n"); scanf("%d",&choice);

switch(choice){

case 1:printf("enter value to be inserted:"); scanf("%d",&value);

push(value); break;

case 2:pop(); break;

case 3:display(); break;

default: printf("invalid input");

}

}

}

void push(int value){ if(top==max-1){

printf("stack is full");

}

else{

stack[top]=value; top=top+1; printf("value inserted");

}

}

void pop(){ if(top==-1){

printf("stack is empty");

}

else{

printf("the deleted element is : %d",stack[top]); top=top-1;

}

}

void display(){ if(top==-1)

printf("no elements in stack to display"); else{

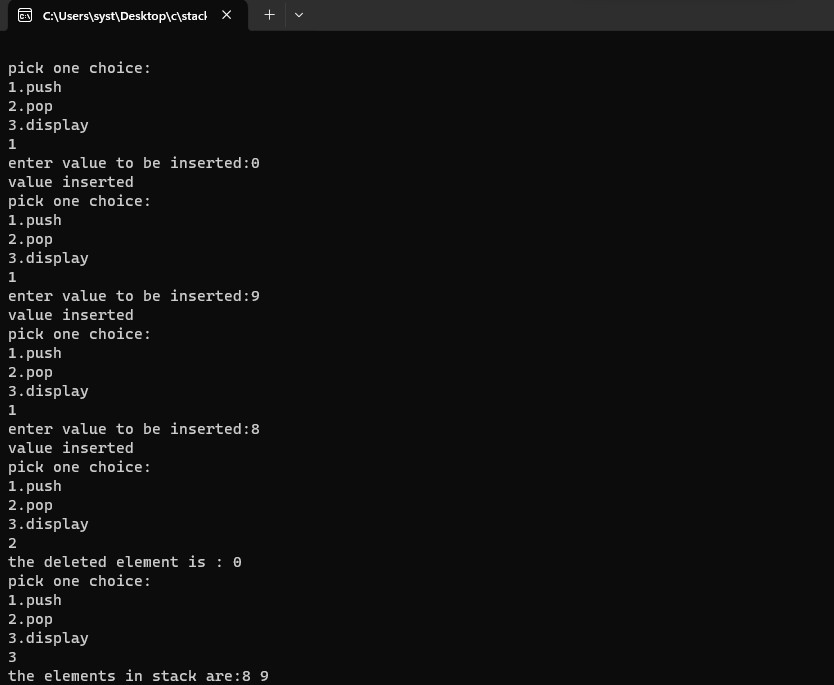
printf("the elements in stack are:"); for (int i=top;i>=0;i--)

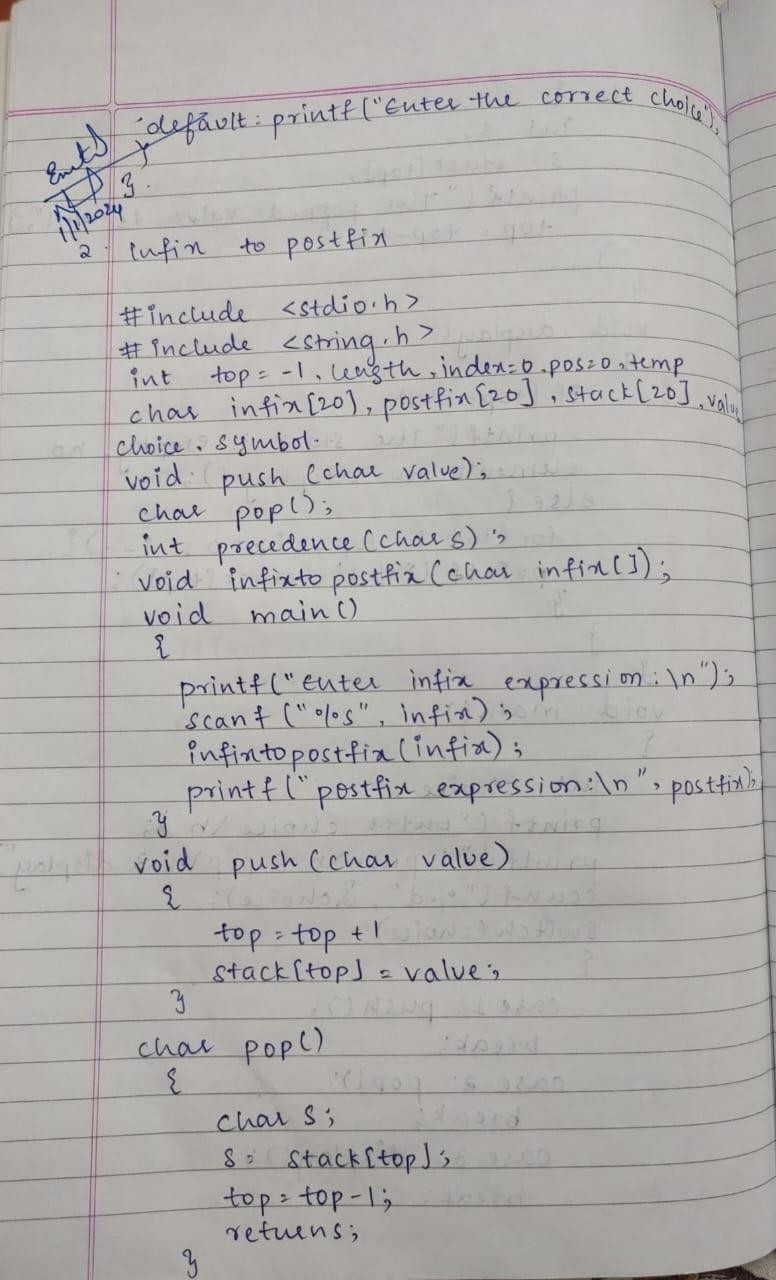
{

printf("%d ",stack[i]);

}

}

}



1. **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 <string.h>

char postfix[100], symbol, stack[100]; int pos = 0, index = 0, top = -1, length; char infix[100];

void push(char value); char pop();

int precedence(char choice); void infixtopostfix();

int main()

{

printf("Enter infix expression:\n"); scanf("%s", infix);

infixtopostfix();

printf("Infix expression: %s\n", infix); printf("Postfix expression: %s\n", postfix); return 0;

}

void push(char value)

{

top = top + 1; stack[top] = value;

}

char pop()

{

char s;

s = stack[top]; top = top - 1; return s;

}

int precedence(char choice)

{

int p;

switch (choice)

{

case '^': p = 3;

break; case '/':

case '\*': p = 2;

break; case '-':

case '+': p = 1;

break; case '(':

p = 0;

break; case '#': p = -1;

break;

}

return p;

}

void infixtopostfix()

{

length = strlen(infix); while (length > index)

{

symbol = infix[index]; switch (symbol)

{

case '(':

push(symbol); break;

case ')':

while (stack[top] != '(')

{

postfix[pos] = pop(); pos++;

}

pop(); break; case '+':

case '-':

case '/':

case '^':

case '\*':

while (precedence(stack[top]) >= precedence(symbol))

{

postfix[pos] = pop();

pos++;

}

push(symbol); break;

default:

postfix[pos] = symbol; pos++;

}

index++;

}

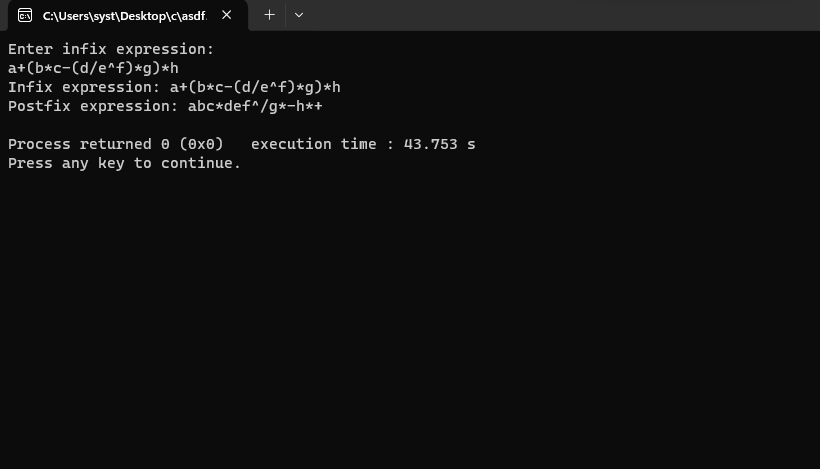
while (top != -1)

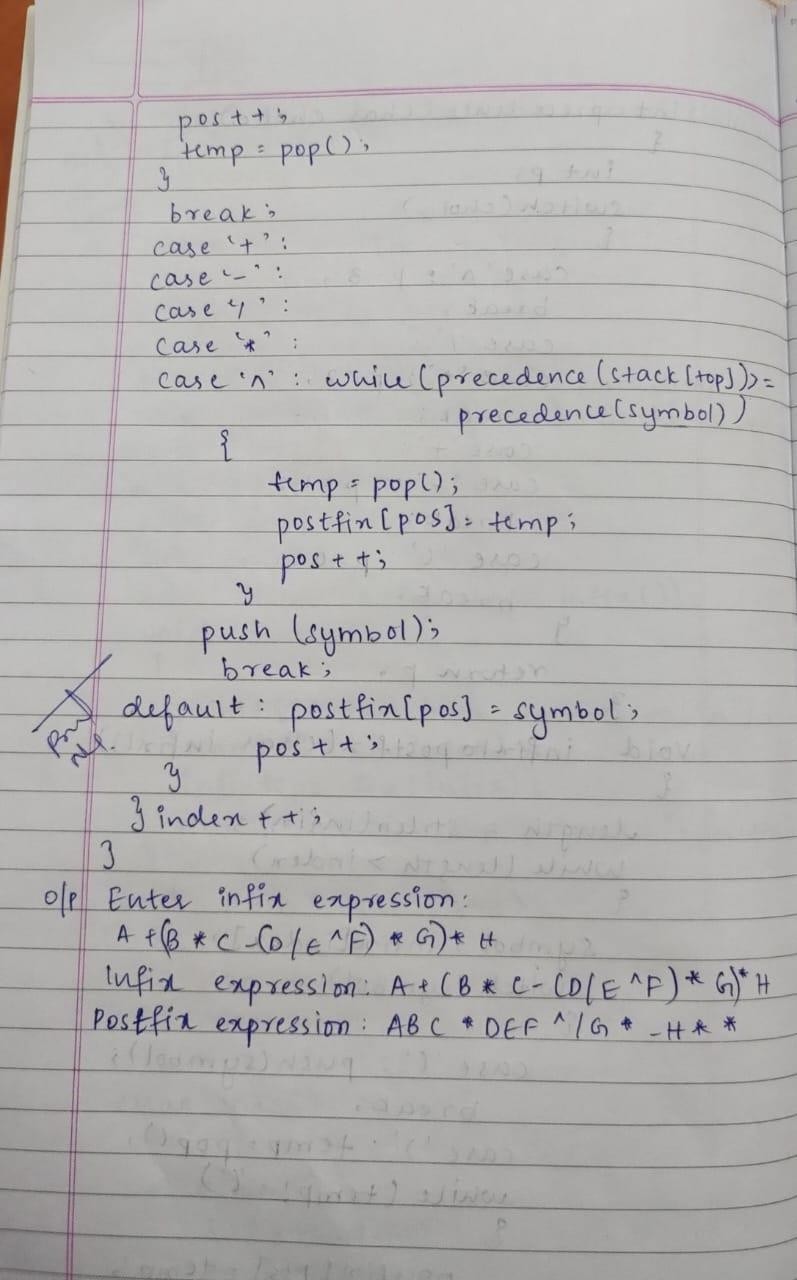
{

postfix[pos] = pop(); pos++;

}

postfix[pos] = '\0';

}



1. **write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate message for overflow and underflow condition.**

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

#define max 5

void display();

void insert(int value); int deletion();

int value, q[max], front = -1, rear = -1;

void main() { int choice; while (1) {

printf("\nenter 1.insert 2.delete 3.display 4.exit"); scanf("%d", &choice);

switch (choice) { case 1:

printf("enter value:"); scanf("%d", &value); insert(value); printf("\nvalue inserted\n"); break;

case 2:

deletion(); break;

case 3:

display(); break;

case 4:

exit(0); default:

printf("invalid input");

}

}

}

void display() { if (front == -1)

printf("underflow"); else {

printf("the elements in queue are:\n"); for (int i = front; i <= rear; i++) {

printf("%d ", q[i]);

}

}

}

void insert(int value) { if (rear == max - 1)

printf("overflow"); else {

if (front == -1) { front = 0;

}

rear = rear + 1; q[rear] = value;

}

}

int deletion() {

if (front == -1) { printf("underflow\n");

} else {

int s = q[front]; front = front + 1;

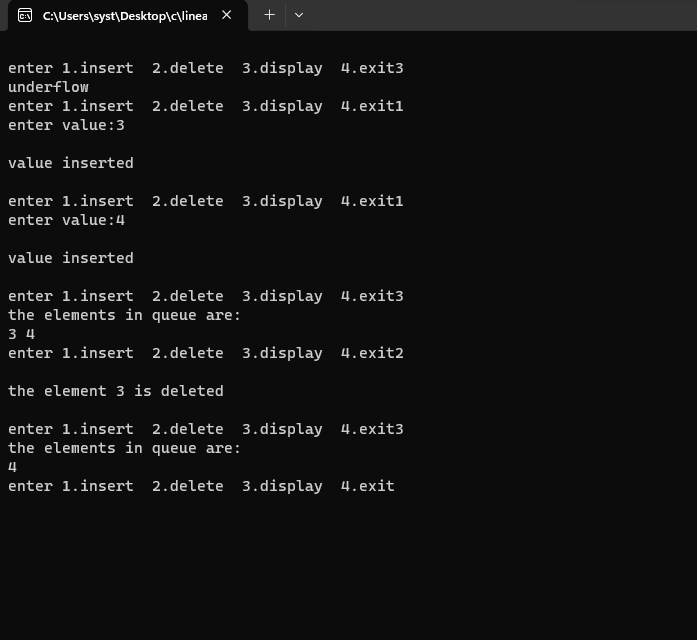
if (front > rear) { front = rear = -1;

}

printf("\nthe element %d is deleted\n", s); return s;

}

}



1. **write a program to simulate the working of a circular queue using an array. Provide the following operations: insert, delete& display. The program should print appropriate message for queue empty and queue overflow conditions.**

#include <stdio.h> #define size 3

int isfull();

int isempty(); int deque();

void enque(int value); void display();

int front=-1,rear=-1,q[size],value,s; void main()

{

int choice; while(1){

printf("enter 1.insert 2.delete 3.display 4.exit"); scanf("%d", &choice);

switch (choice) { case 1:

printf("enter value:"); scanf("%d", &value); enque(value);

break; case 2:

deque(); break;

case 3:

display(); break;

case 4:

exit(0); default:

printf("invalid input");

}

}

}

int isfull(){ if((front==rear+1)||(front==0&&rear==size-1))

return 1; else

return 0;}

int isempty()

{

if(front==-1) return 1;

else

return 0;

}

void enque(int value)

{

if (isfull()) printf("overflow\n");

else

{

if(front==-1){ front=0;

}

rear=(rear+1)%size; q[rear]=value; printf("\nvalue inserted\n");

}

}

int deque()

{

if (isempty()) printf("underflow\n");

else

{

s=q[front];

if(front==rear) front=rear=-1;

else

{

front=(front+1)%size; return s;

}

}

}

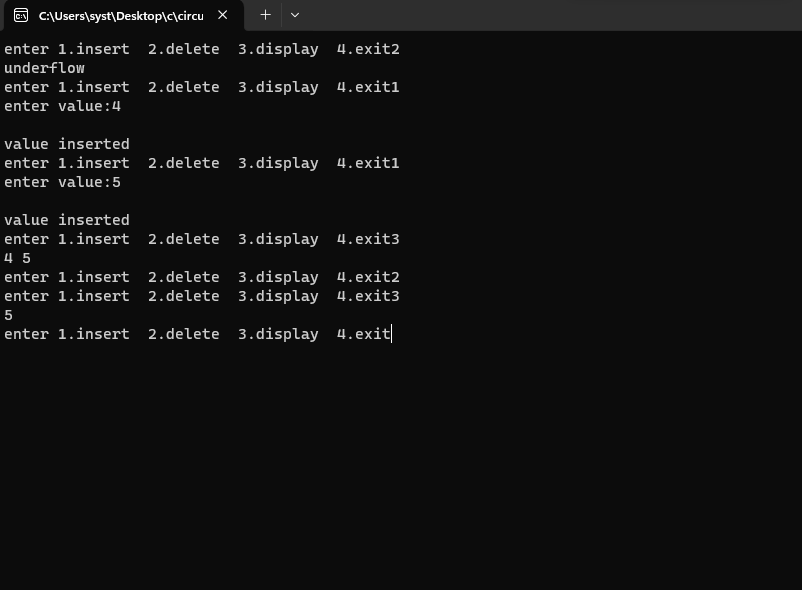
void display() { if (isempty())

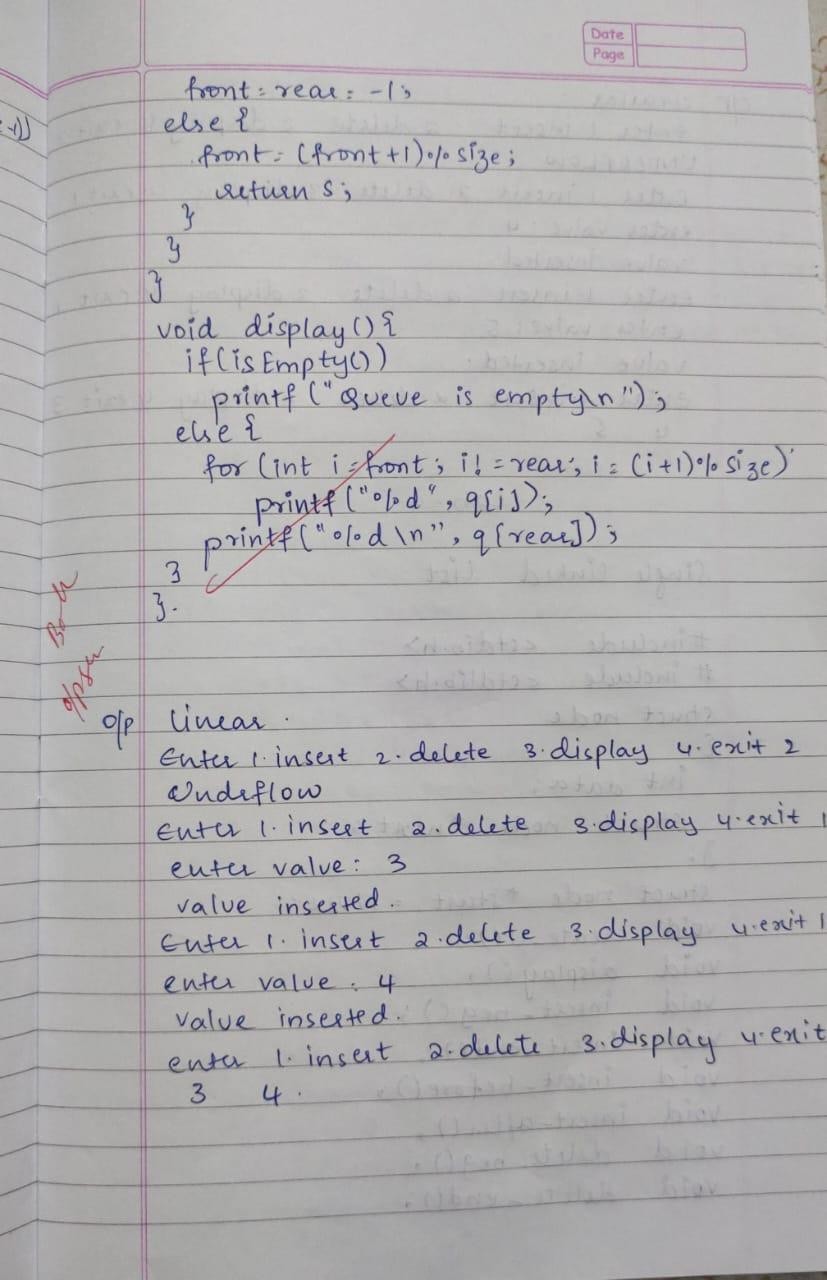
printf("underflow\n"); else {

for (int i = front; i != rear; i = (i + 1) % size) { printf("%d ", q[i]);

}

printf("%d\n", q[rear]);

}



1. **WAP to Implement Singly Linked List with following operations.**
   1. **Create a linked list.**
   2. **Insertion of a node at first position, at any position and at end of list.**
   3. **Display the contents of the linked list.**
2. **WAP to Implement Singly Linked List with following operations.**
   1. **Create a linked list.**
   2. **Deletion of first element, specified element and last element in the list.**
   3. **Display the contents of the linked list.**

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

struct node

{

int data;

struct node \*next;

};

struct node \*start = NULL;

void create\_ll(); void display(); void insert\_beg(); void insert\_end();

void insert\_before(); void insert\_after(); void delete\_beg(); void delete\_end(); void delete\_node(); void exit\_program();

int main()

{

int choice;

printf("\n1. Create linked list\n"); printf("2. Display\n");

printf("3. Insert at the beginning\n"); printf("4. Insert at the end\n");

printf("5. Insert before a given value\n"); printf("6. Insert after a given value\n"); printf("7. Delete from the beginning\n"); printf("8. Delete from the end\n"); printf("9. Delete a specific node\n"); printf("10. Exit\n");

while (1)

{

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

switch (choice)

{

case 1:

create\_ll(); break;

case 2:

display(); break;

case 3:

insert\_beg(); break;

case 4:

insert\_end();

break; case 5:

insert\_before(); break;

case 6:

insert\_after(); break;

case 7:

delete\_beg(); break;

case 8:

delete\_end(); break;

case 9:

delete\_node(); break;

case 10: exit\_program(); break;

default:

printf("Invalid input\n");

}

}

return 0;

}

void create\_ll()

{

int option, num;

struct node \*ptr, \*newnode;

while (1)

{

printf("\nEnter 1. Creating list 2. Exit\n"); scanf("%d", &option);

switch (option)

{

case 1:

newnode = (struct node \*)malloc(sizeof(struct node));

printf("Enter the value to be inserted: "); scanf("%d", &num);

newnode->data = num; newnode->next = NULL;

if (start == NULL)

{

start = newnode;

}

else

{

ptr = start;

while (ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = newnode;

}

break;

case 2:

return; default:

printf("Invalid input\n");

}

}

}

void display()

{

if (start == NULL)

{

printf("\nList is empty\n"); return;

}

struct node \*ptr; ptr = start;

printf("\nLinked List: "); while (ptr != NULL)

{

printf("%d\t", ptr->data); ptr = ptr->next;

}

printf("\n");

}

void insert\_beg()

{

struct node \*newnode;

int num;

printf("\nEnter the value to be inserted: "); scanf("%d", &num);

newnode = (struct node \*)malloc(sizeof(struct node)); newnode->data = num;

newnode->next = start; start = newnode;

}

void insert\_end()

{

struct node \*newnode, \*ptr; int num;

printf("\nEnter the value to be inserted: "); scanf("%d", &num);

newnode = (struct node \*)malloc(sizeof(struct node)); newnode->data = num;

newnode->next = NULL;

ptr = start;

if (start == NULL)

{

start = newnode;

}

else

{

while (ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = newnode;

}

}

void insert\_before()

{

struct node \*newnode, \*ptr, \*preptr; int num, value;

printf("\nEnter the value to be inserted: "); scanf("%d", &num);

printf("Enter the value before which the data should be inserted: "); scanf("%d", &value);

newnode = (struct node \*)malloc(sizeof(struct node)); ptr = start;

newnode->data = num;

while (ptr->data != value)

{

preptr = ptr; ptr = ptr->next;

}

preptr->next = newnode; newnode->next = ptr;

}

void insert\_after()

{

struct node \*newnode, \*ptr; int num, value;

printf("\nEnter the value to be inserted: "); scanf("%d", &num);

printf("Enter the value after which the data should be inserted: "); scanf("%d", &value);

newnode = (struct node \*)malloc(sizeof(struct node)); ptr = start;

newnode->data = num;

while (ptr->data != value)

{

ptr = ptr->next;

}

newnode->next = ptr->next; ptr->next = newnode;

}

void delete\_beg()

{

if (start == NULL)

{

printf("\nList is empty. Nothing to delete.\n"); return;

}

struct node \*ptr; ptr = start;

start = start->next; free(ptr);

printf("\nNode deleted from the beginning\n");

}

void delete\_end()

{

if (start == NULL)

{

printf("\nList is empty. Nothing to delete.\n"); return;

}

struct node \*ptr, \*preptr; ptr = start;

while (ptr->next != NULL)

{

preptr = ptr; ptr = ptr->next;

}

preptr->next = NULL; free(ptr);

printf("\nNode deleted from the end\n");

}

void delete\_node()

{

if (start == NULL)

{

printf("\nList is empty. Nothing to delete.\n"); return;

}

struct node \*ptr, \*preptr; int value;

printf("\nEnter value to be deleted: "); scanf("%d", &value);

ptr = start;

if (ptr->data == value)

{

start = start->next; free(ptr);

printf("\nNode with value %d deleted\n", value); return;

}

while (ptr != NULL && ptr->data != value)

{

preptr = ptr; ptr = ptr->next;

}

if (ptr == NULL)

{

printf("\nNode with value %d not found\n", value);

return;

}

preptr->next = ptr->next; free(ptr);

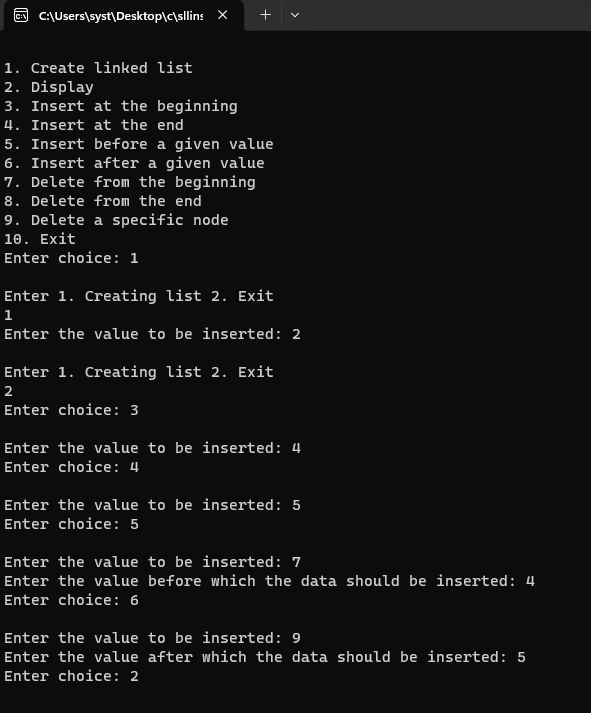
printf("\nNode with value %d deleted\n", value);

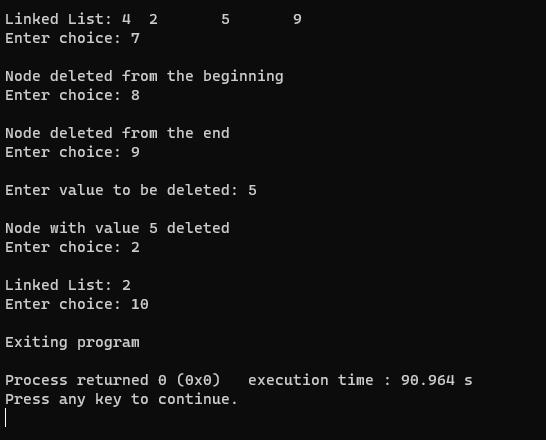
}

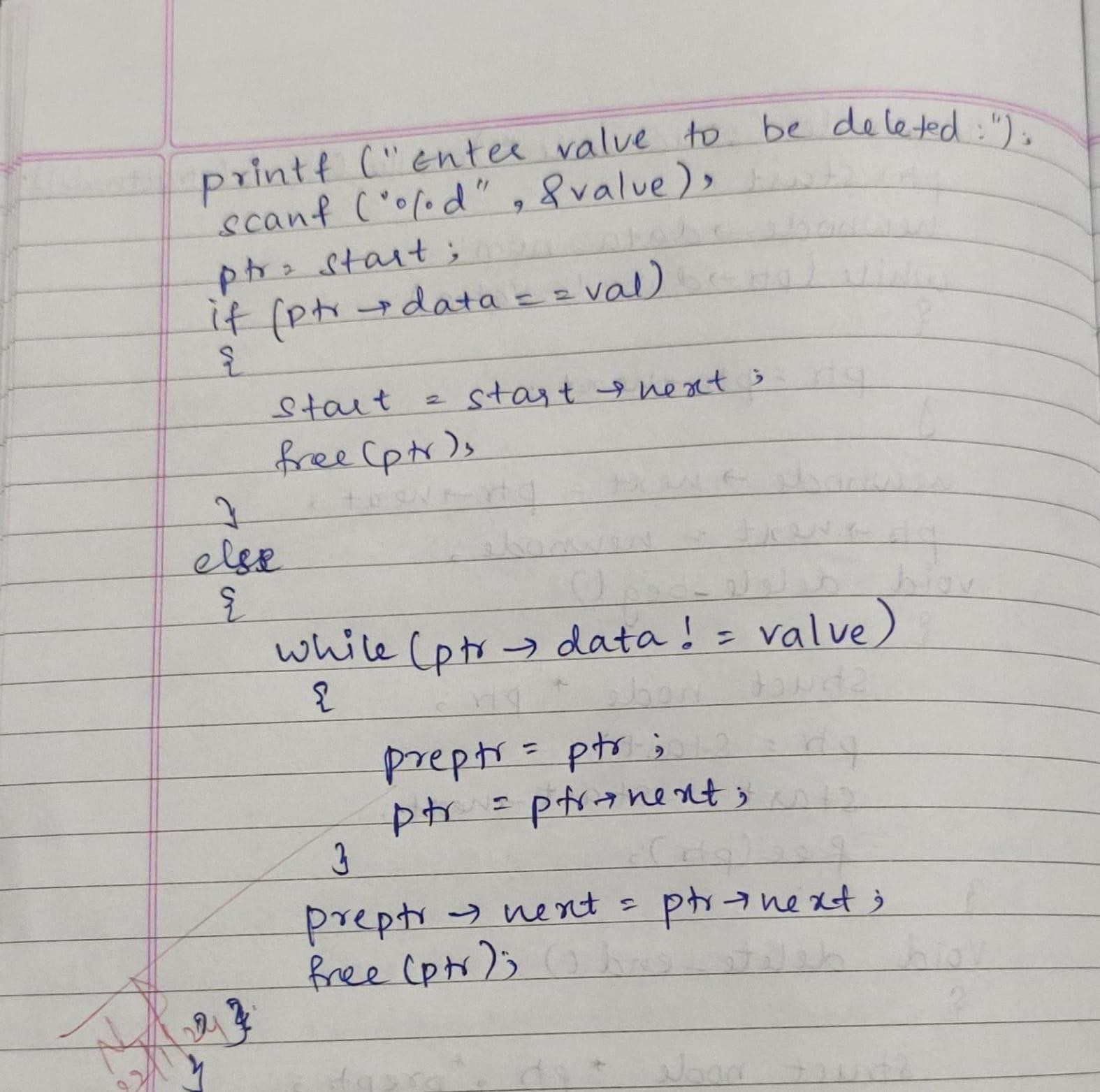
void exit\_program()

{

printf("\nExiting program\n"); exit(0);

}





1. **Single linked list -sort,reverse,concatination Sort,reverse linked list**

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

{

int data;

struct node \*next;

};

struct node \*start=NULL;

void create\_ll(); void sort\_ll(); void display(); void reverse(); void main()

{

printf("1.create linked list\n2.sort linked list\n3.display\n4.reverse the list\n5.exit"); int choice;

while(1)

{

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

{

case 1:create\_ll(); break;

case 2:sort\_ll(); break;

case 3:display(); break;

case 4:reverse(); break;

case 5:exit(0); break;

default:printf("Invalid input\n");

}

}

}

void create\_ll()

{

int option, num;

struct node \*ptr, \*newnode;

while (1)

{

printf("\nEnter 1. Creating list 2. Exit\n"); scanf("%d", &option);

switch (option)

{

case 1:

newnode = (struct node \*)malloc(sizeof(struct node));

printf("Enter the value to be inserted: "); scanf("%d", &num);

newnode->data = num; newnode->next = NULL;

if (start == NULL)

{

start = newnode;

}

else

{

ptr = start;

while (ptr->next != NULL)

{

ptr = ptr->next;

}

ptr->next = newnode;

}

break; case 2:

return; default:

printf("Invalid input\n");

}

}

}

void display()

{

if (start == NULL)

{

printf("\nList is empty\n"); return;

}

struct node \*ptr; ptr=start; while(ptr!=NULL)

{

printf("%d ",ptr->data); ptr=ptr->next;

}

}

void sort\_ll()

{

struct node \*ptr1,\*ptr2; int temp;

ptr1=start;

while(ptr1->next!=NULL)

{

ptr2=ptr1->next; while(ptr2!=NULL)

{

if(ptr1->data > ptr2->data)

{

temp=ptr1->data;

ptr1->data=ptr2->data; ptr2->data=temp;

}

ptr2=ptr2->next;

}

ptr1=ptr1->next;

}

}

void reverse()

{

struct node \*prev=NULL,\*temp=NULL,\*current=start; while(current!=NULL)

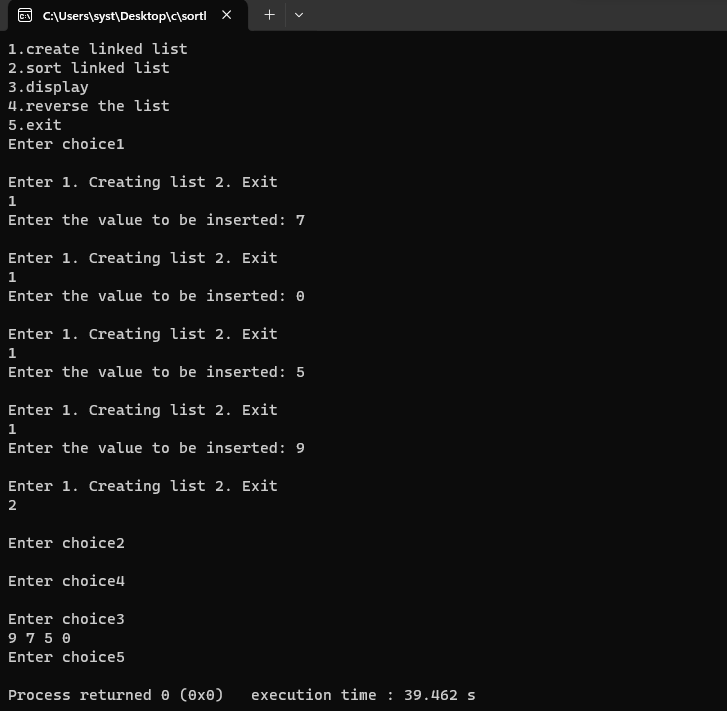
{

temp=current->next; current->next=prev; prev=current; current=temp;

}

start=prev;

}



**Concatenate two lists**

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

struct node

{

int data;

struct node \*next;

};

struct node \*head = NULL;

void display(struct node \*current)

{

if (current == NULL)

{

printf("NULL\n");

}

else

{

printf("%d --> ", current->data); display(current->next);

}

}

void concatenate(struct node \*a, struct node \*b)

{

if (a != NULL && b != NULL)

{

if (a->next == NULL) a->next = b;

else

concatenate(a->next, b);

}

else

{

printf("Either a or b is NULL\n");

}

}

int main()

{

struct node \*prev, \*a, \*b, \*p;

int n, i;

printf("Number of elements in a: "); scanf("%d", &n);

printf("Enter the elements in a\n"); a = NULL;

for (i = 0; i < n; i++)

{

p = malloc(sizeof(struct node)); scanf("%d", &p->data);

p->next = NULL; if (a == NULL)

a = p; else

prev->next = p; prev = p;

}

printf("Number of elements in b: "); scanf("%d", &n);

b = NULL;

printf("Enter the elements in b\n"); for (i = 0; i < n; i++)

{

p = malloc(sizeof(struct node)); scanf("%d", &p->data);

p->next = NULL; if (b == NULL)

b = p; else

prev->next = p; prev = p;

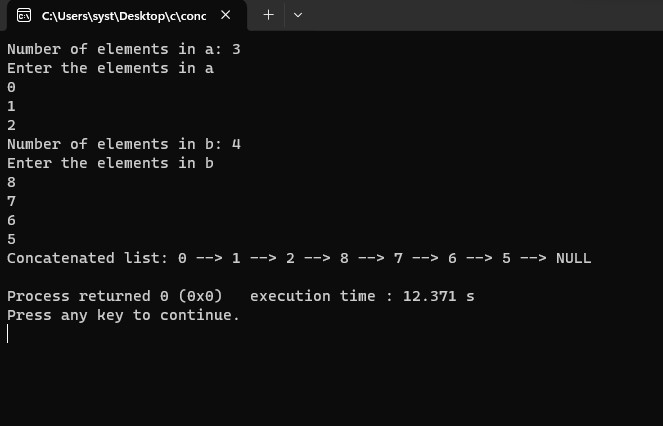
}

concatenate(a, b);

printf("Concatenated list: "); display(a);

return 0;

}



**8a.Stack implementation using single linked list**

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

{

int data;

struct node \*next;

}\*top,\*temp;

void push(int value); int pop();

void display(); void main()

{

int choice,value;

printf("Implementation of stack using single linked list\n"); printf("1.push\t2.pop\t3.display\t4.exit\n");

while(1)

{

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

{

case 1:printf("Enter the value to be inserted:"); scanf("%d",&value);

push(value); break;

case 2:printf("The popped value is %d\n",pop()); break;

case 3:display(); break;

case 4:exit(0); break;

default:printf("Invalid input");

}

}

}

void push(int value)

{

if (top == NULL)

{

top = (struct node \*)malloc(sizeof(struct node)); top->data = value;

top->next = NULL;

}

else

{

temp = (struct node \*)malloc(sizeof(struct node)); temp->data = value;

temp->next = top; top = temp;

}

printf("Node inserted\n");

}

int pop()

{

temp=top; if(top==NULL)

{

printf("Stack underflow"); return -1;

}

else{

temp=temp->next;

}

int popped=top->data; free(top);

top=temp; return popped;

}

void display()

{

temp = top;

printf("The elements in stack are\n"); if (top == NULL)

{

printf("Stack underflow\n"); return;

}

while (temp != NULL)

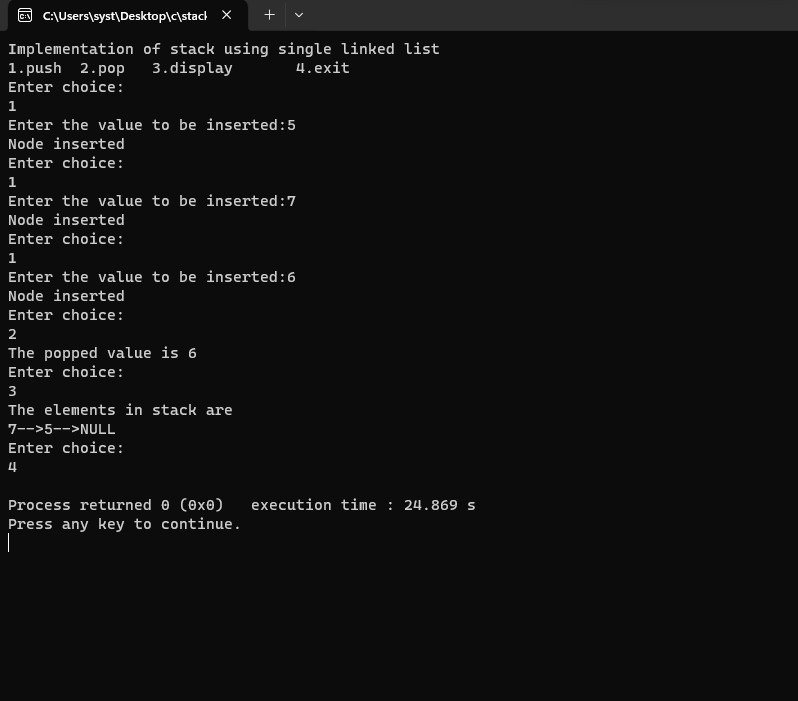
{

printf("%d-->", temp->data); temp = temp->next;

}

printf("NULL\n");

}



**8b.Queue implementation using single linked list**

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

{

int data;

struct node \*next;

};

struct node \*front=NULL,\*rear=NULL; void enqueue(int value);

int dequeue(); void display(); void main()

{

int choice,value;

printf("Implementation of queue using single linked list\n"); printf("\n1.enqueue\t2.dequeue\t3.display\t4.exit\n"); while(1)

{

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

{

case 1:printf("Enter the value to be inserted:"); scanf("%d",&value);

enqueue(value); break;

case 2:printf("The popped value is %d\n",dequeue()); break;

case 3:display(); break;

case 4:exit(0); break;

default:printf("Invalid input");

}

}

}

void enqueue(int value)

{

struct node \*ptr=(struct node \*)malloc(sizeof(struct node)); ptr->data=value;

ptr->next=NULL;

if(front==NULL && rear==NULL)

{

front=rear=ptr;

}

else{

rear->next=ptr; rear=ptr;

}

printf("Node inserted\n");

}

int dequeue()

{

if(front==NULL)

{

printf("Queue is empty"); return -1;

}

else{

struct node \*temp=front; int temp\_data=front->data; front=front->next; free(temp);

return temp\_data;

}

}

void display()

{

if(front==NULL)

{

printf("Queue is empty");

}

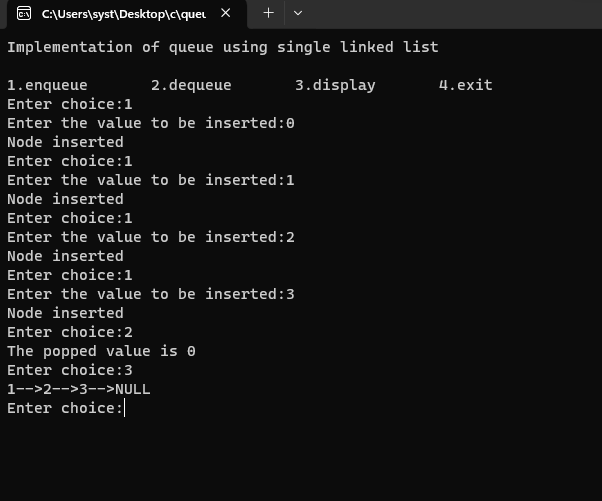
struct node \*temp=front; while(temp!=NULL)

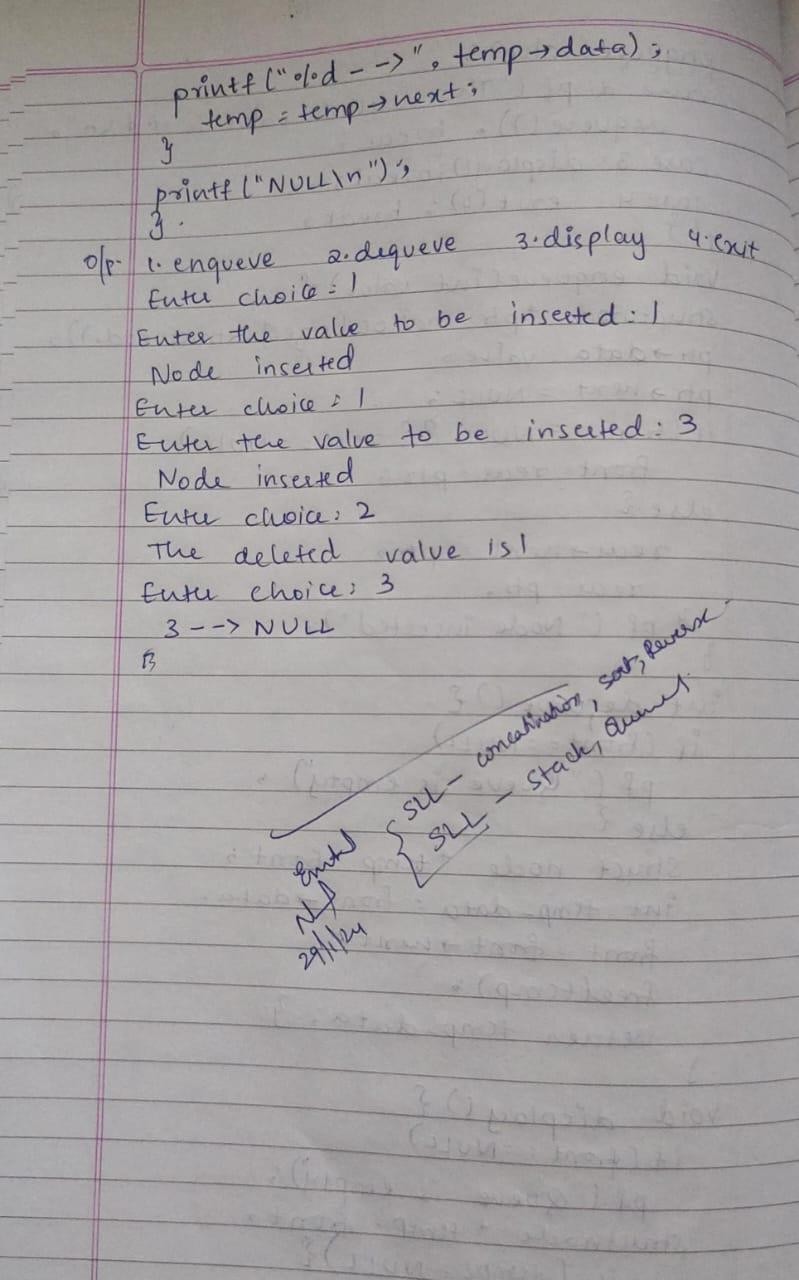
{

printf("%d-->",temp->data); temp=temp->next;

}

printf("NULL\n");

}



1. **WAP to Implement doubly link list with primitive operations**
   1. **Create a doubly linked list.**
   2. **Insert a new node to the left of the node.**
   3. **Delete the node based on a specific value**

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

struct node

{

int data;

struct node \*prev; struct node \*next;

};

struct node \*head = NULL, \*ptr, \*temp, \*newnode;

void create\_ll();

void insert\_before(); void delete\_node(); void display();

void create\_ll()

{

int value;

newnode = (struct node \*)malloc(sizeof(struct node)); printf("Enter the value to be inserted:");

scanf("%d", &value); if (head == NULL)

{

newnode->data = value; newnode->prev = NULL;

newnode->next = NULL; head = newnode;

}

else

{

ptr = head;

while (ptr->next != NULL) ptr = ptr->next;

newnode->data = value; newnode->prev = ptr; newnode->next = NULL; ptr->next = newnode;

}

}

void insert\_before()

{

int value, num;

newnode = (struct node \*)malloc(sizeof(struct node)); printf("Enter the value to be inserted:");

scanf("%d", &value);

printf("Enter before which the value should be inserted:"); scanf("%d", &num);

newnode->data = value;

if (head == NULL || num == head->data)

{

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

{

head->prev = newnode;

}

head = newnode; return;

}

ptr = head;

while (ptr->data != num) ptr = ptr->next;

newnode->prev = ptr->prev; newnode->next = ptr;

ptr->prev->next = newnode; ptr->prev = newnode;

}

void delete\_node()

{

int value; ptr = head;

printf("Enter the value to be deleted:"); scanf("%d", &value);

if (head == NULL)

{

printf("List is empty"); return;

}

if (value == head->data)

{

temp = head;

head = head->next;

head->prev = NULL; free(temp);

return;

}

while(ptr->data!=value){ temp=ptr;

ptr=ptr->next;

}

temp->next=ptr->next; ptr->next->prev=temp; free(ptr);

if (ptr == NULL)

{

printf("Value not found in the list");

}

}

void display()

{

if (head == NULL)

{

printf("List is empty");

}

else

{

ptr = head;

while (ptr != NULL)

{

printf("%d ", ptr->data);

ptr = ptr->next;

}

}

}

int main()

{

int choice;

printf("1. create list 2. insert left 3. delete node 4. display 5. exit "); while (1)

{

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

{

case 1:

create\_ll(); break;

case 2:

insert\_before(); break;

case 3:

delete\_node(); break;

case 4:

display(); break;

case 5:

exit(0); break;

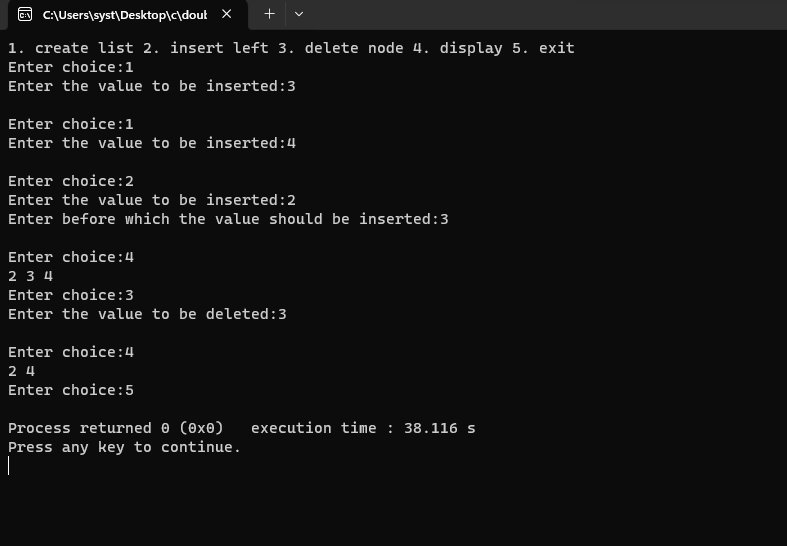
default:

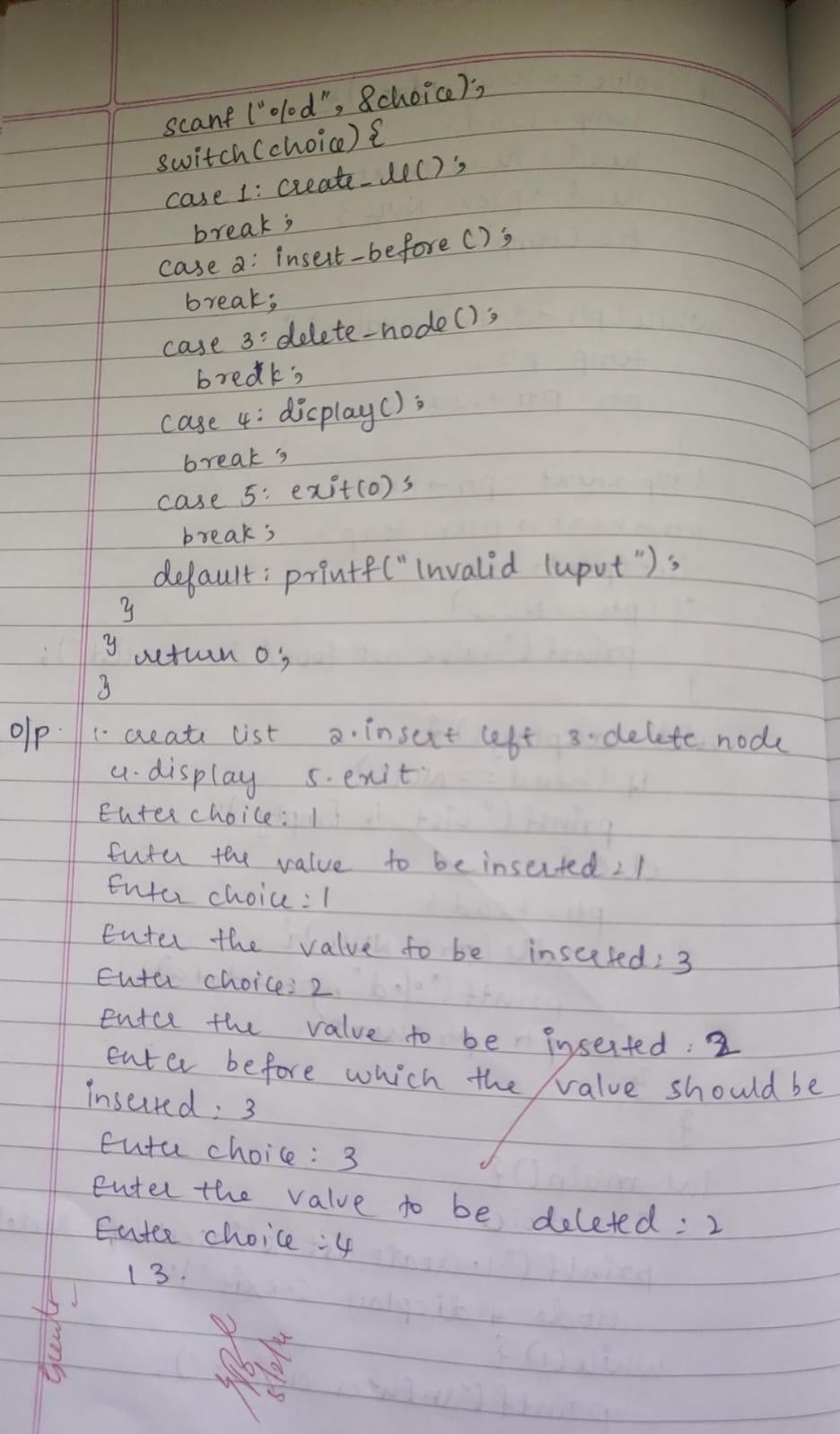
printf("Invalid input!");

}

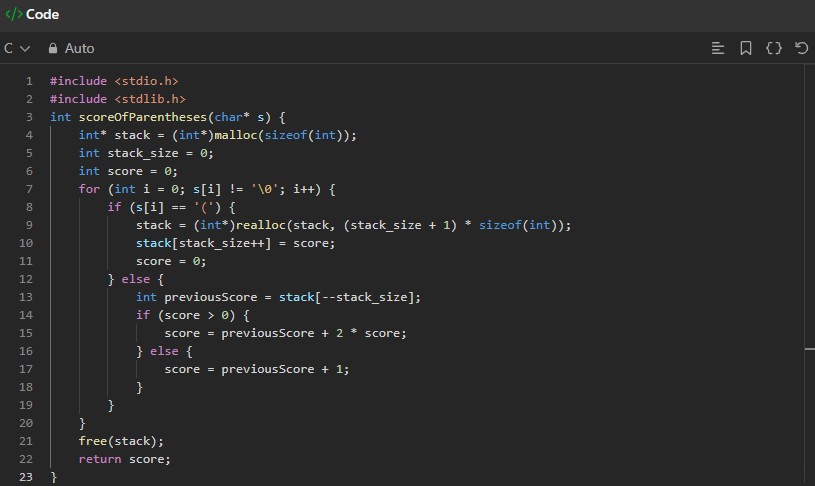
}

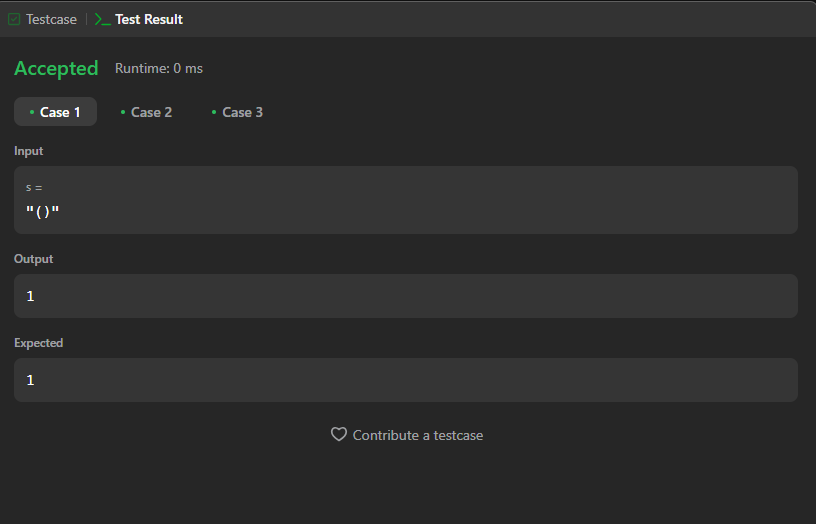
return 0;

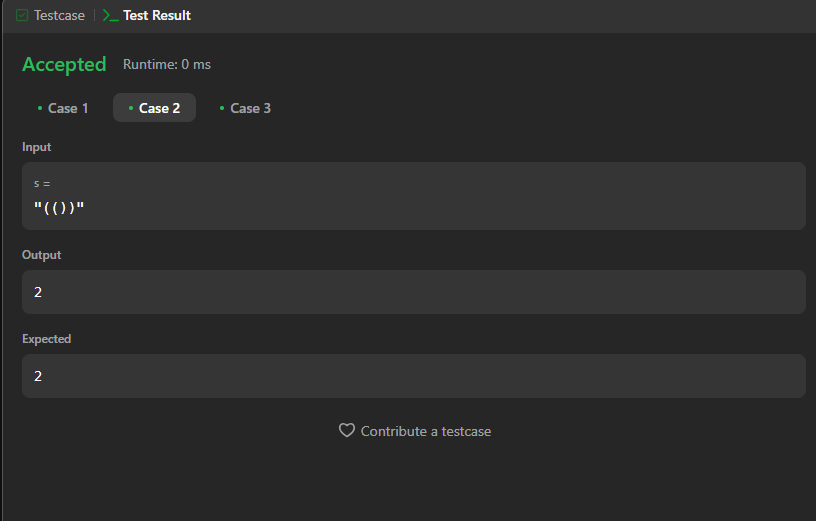
}

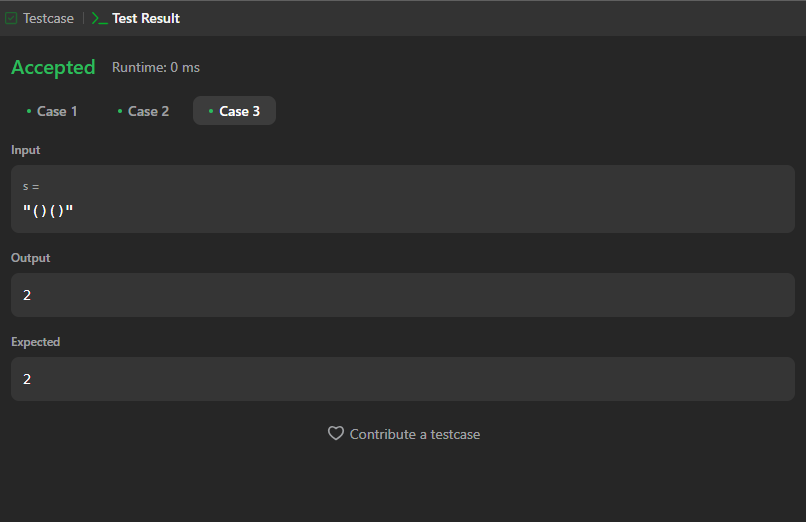


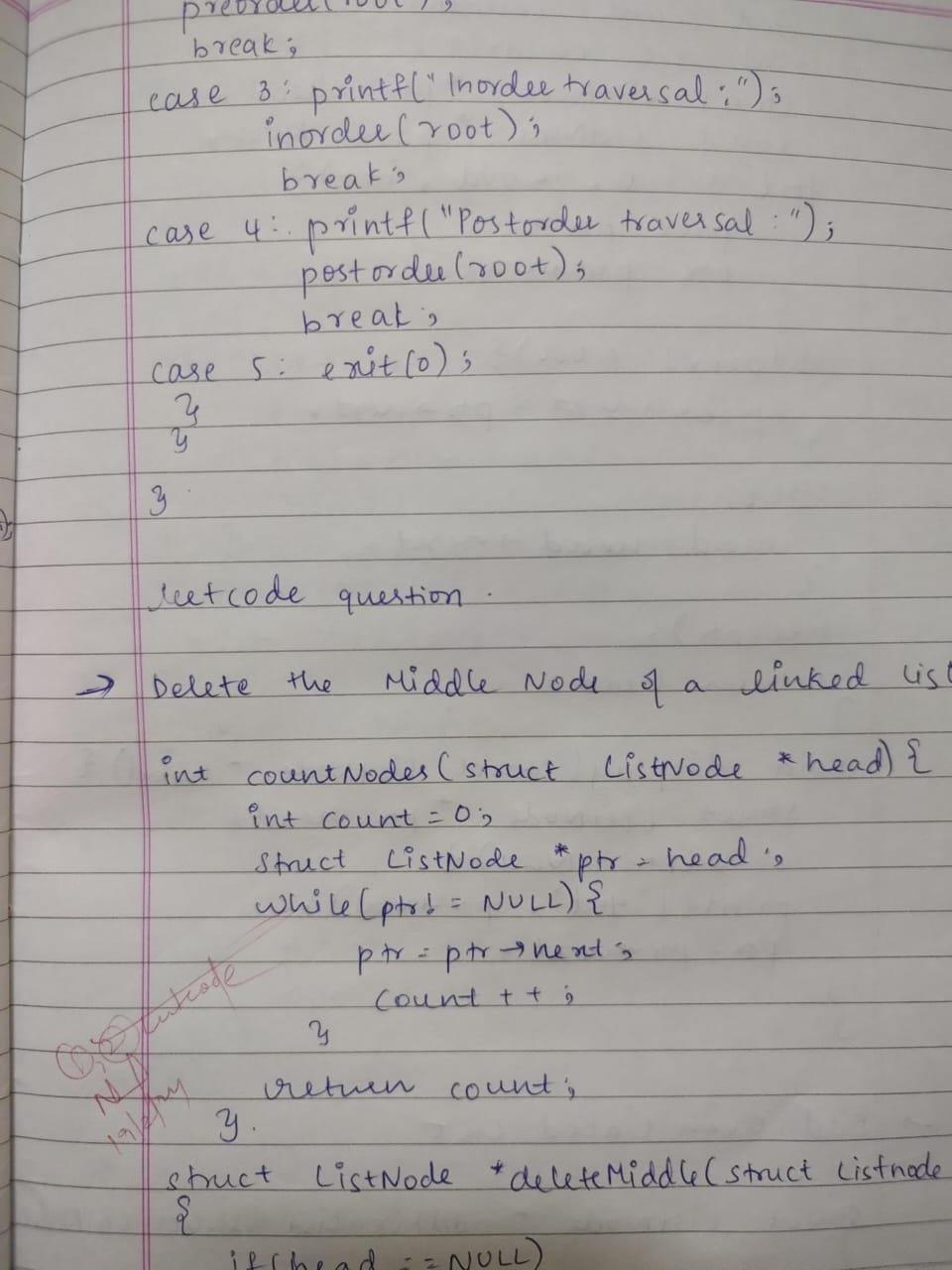
**LEET CODE QUESTION: 856:SCORE OF PARENTHESES**











1. **Write a program.**
2. **To construct Binary Search tree**
3. **Traverse the tree using inorder , postorder, preorder.**
4. **Display the elements in the tree.**

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

struct node { int data;

struct node \*left; struct node \*right;

};

struct node \*create(int value) {

struct node \*newnode = (struct node\*)malloc(sizeof(struct node)); newnode->data = value;

newnode->left = newnode->right = NULL; return newnode;

}

struct node \*insertnode(struct node \*root, int value) { if (root == NULL) {

return create(value);

}

if (value < root->data) {

root->left = insertnode(root->left, value);

}

else if (value > root->data) {

root->right = insertnode(root->right, value);

}

return root;

}

void postorder(struct node \*root) { if (root != NULL) {

postorder(root->left); postorder(root->right); printf("%d ", root->data);

}

}

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 main() {

struct node \*root = NULL; int choice, value;

printf("\n1. Insert\n"); printf("2. Display Preorder\n"); printf("3. Display Inorder\n");

printf("4. Display Postorder\n"); printf("5. Exit\n");

while(1){

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

switch (choice) { case 1:

printf("Enter value to insert: "); scanf("%d", &value);

root = insertnode(root, value); break;

case 2:

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

break; case 3:

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

break; case 4:

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

break; case 5:

exit(0); break;

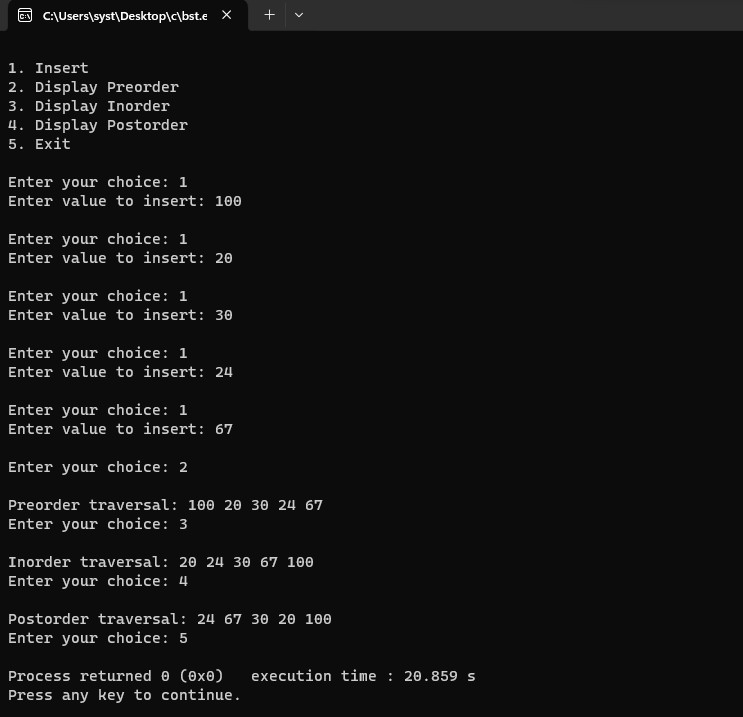
default:

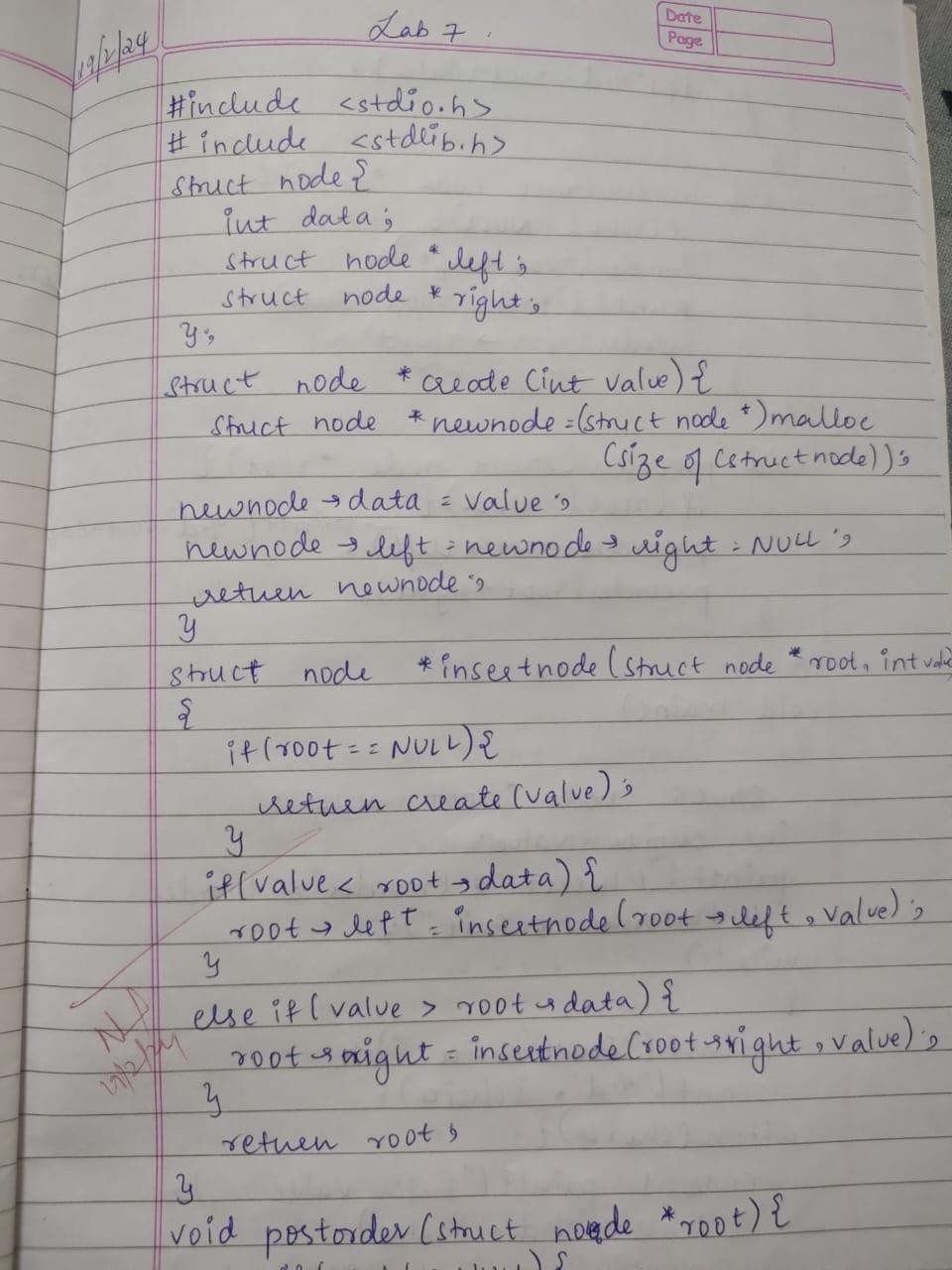
printf("Invalid choice.\n");

}

}

}





**LEET CODE QUESTIONS:**

**Delete the Middle Node of a Linked List**

int countNodes(struct ListNode \*head) { int count = 0;

struct ListNode \*ptr = head; while (ptr != NULL) {

ptr = ptr->next; count++;

}

return count;

}

struct ListNode\* deleteMiddle(struct ListNode\* head) { if (head == NULL)

return NULL;

if (head->next == NULL) { free(head);

return NULL;

}

int count = countNodes(head); int mid = count / 2;

struct ListNode \*ptr = head; struct ListNode \*prev = NULL;

for (int i = 0; i < mid; i++) { prev = ptr;

ptr = ptr->next;

}

if (prev != NULL) {

prev->next = ptr->next;

} else {

head = head->next;

}

free(ptr); return head;

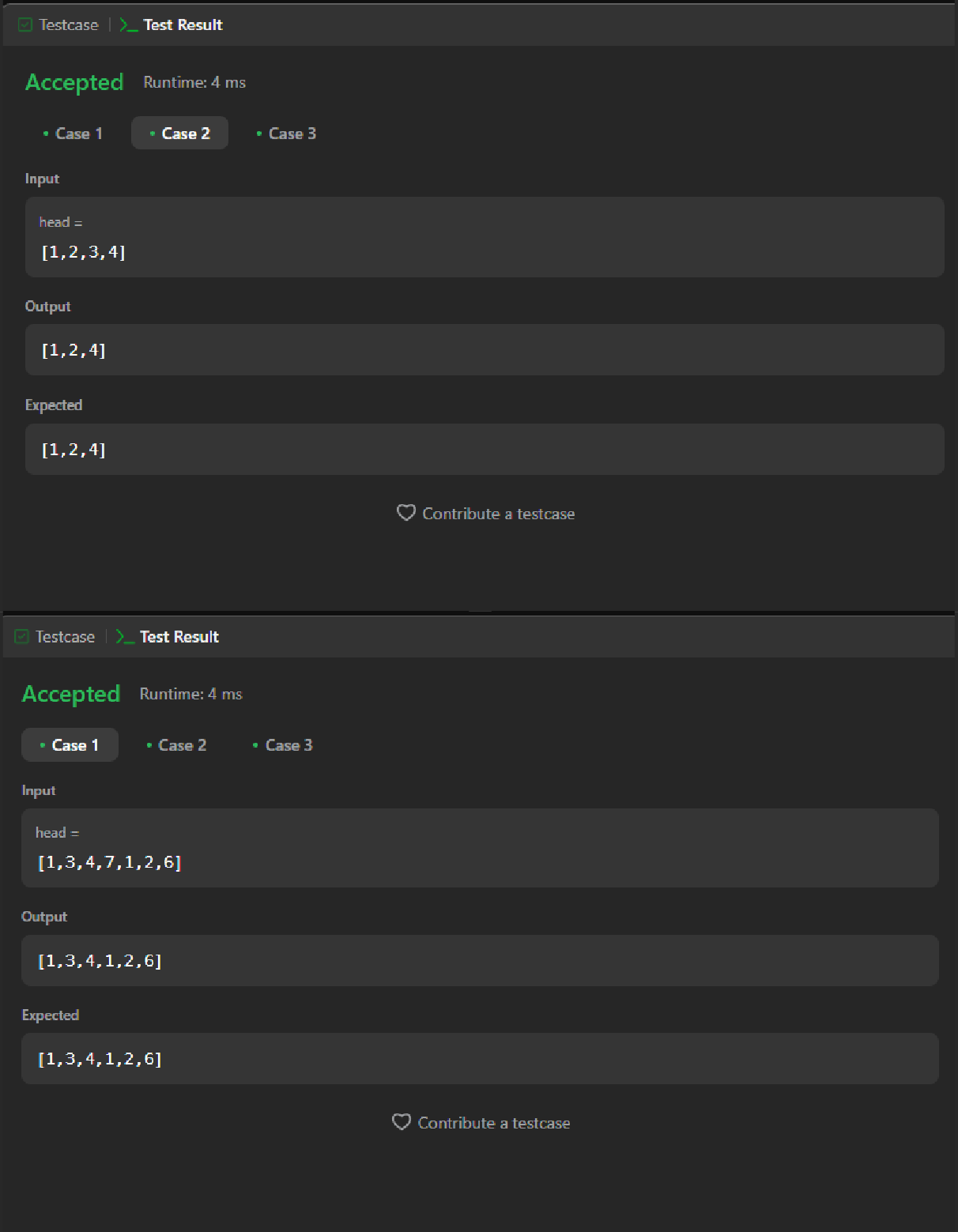
}

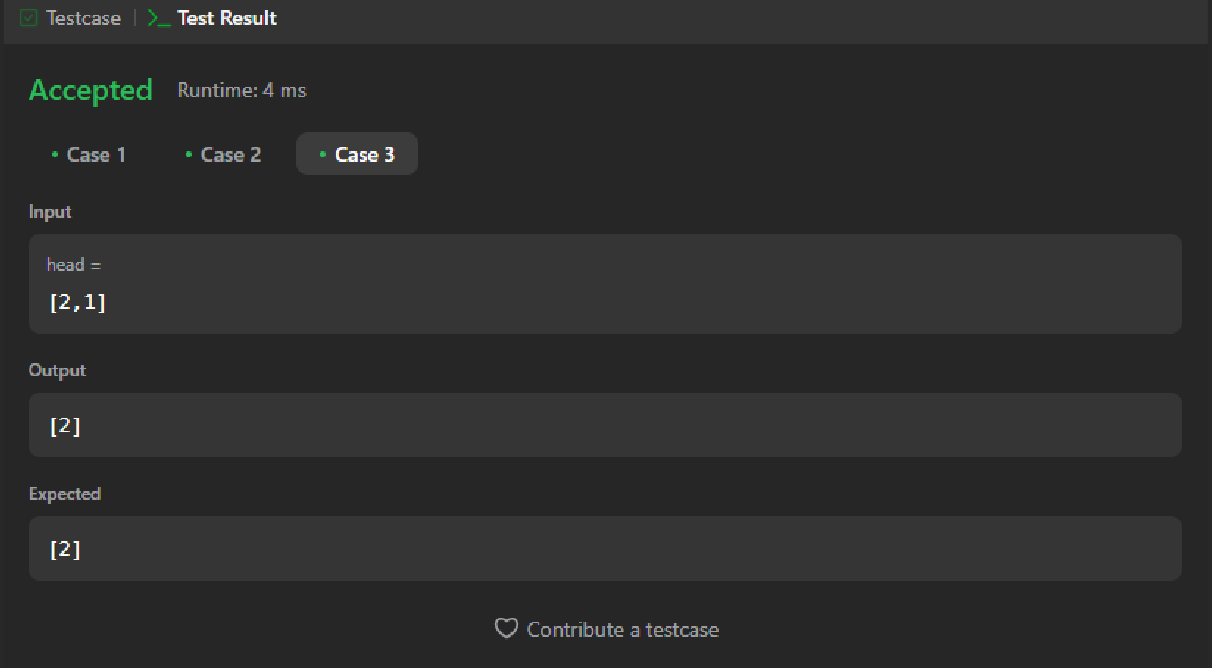
void display(struct ListNode \*head) { struct ListNode \*ptr = head; while (ptr != NULL) {

printf("%d ", ptr->val); ptr = ptr->next;

}

}





**Odd Even Linked List**

struct ListNode\* oddEvenList(struct ListNode\* head) { struct ListNode \*even=head->next;

struct ListNode \*odd=head; struct ListNode \*ptr=even; while(1){

if(!odd || !even || !even->next){ odd->next=ptr;

break;

}

odd->next=even->next; odd=even->next;

if (odd->next == NULL)

{

even->next = NULL; odd->next = ptr; break;

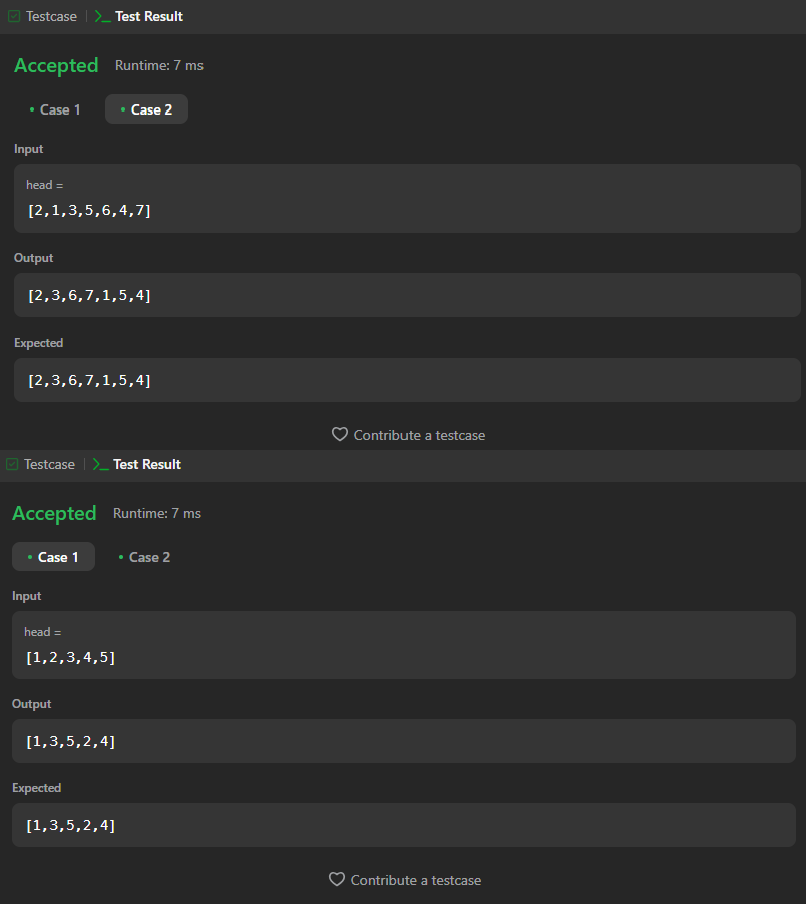
}

even->next = odd->next; even = odd->next;

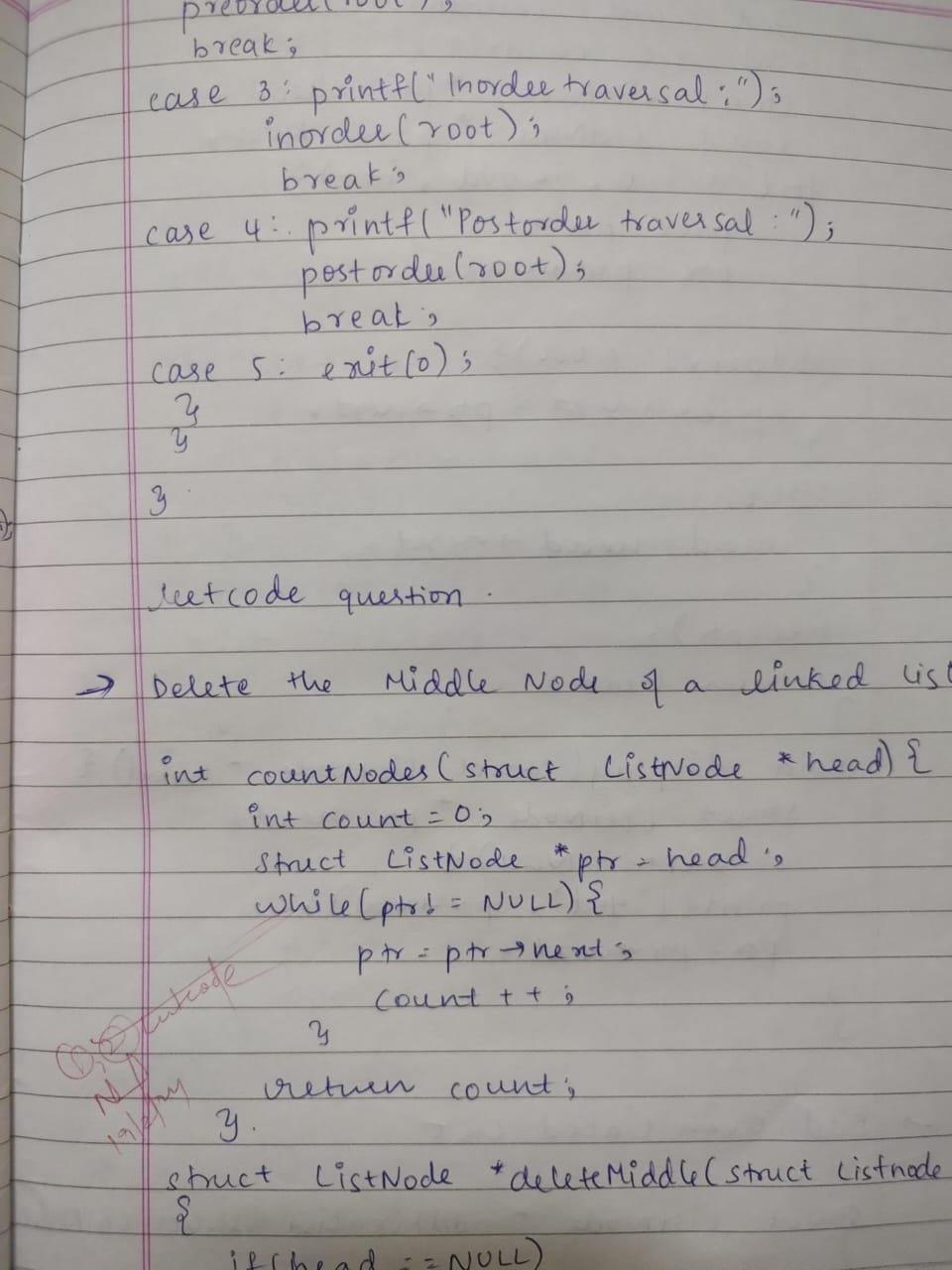
}

return head;

}



}



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

#include <stdio.h> #define max 100

int front=-1,rear=-1; int queue[max];

void enqueue(int value)

{

if(rear==max-1) printf("Overflow");

if(front==-1) front=0; rear=rear+1; queue[rear]=value;

}

int dequeue()

{

if(front==-1){ printf("Underflow"); return -1;

}

int node=queue[front]; front=front+1;

return node;

}

void bfs(int a[max][max],int g[max],int u,int n)

{

int v[max]={0}; enqueue(u); v[u]=1;

while(front<=rear)

{

int cur=dequeue(); printf("%d ",cur); for(int i=0;i<n;i++){

if(a[cur][i]==1 && v[i]==0){ enqueue(g[i]);

v[i]=1;

}

}

}

}

void main()

{

printf("Enter no.of elements in the graph:"); int n,u,g[max],a[max][max]; scanf("%d",&n);

printf("Enter elements in the graph:"); for(int i=0;i<n;i++){

scanf("%d",&g[i]);

}

printf("Enter root element:"); scanf("%d",&u);

printf("Enter adjacency matrix:"); for(int i=0;i<n;i++){

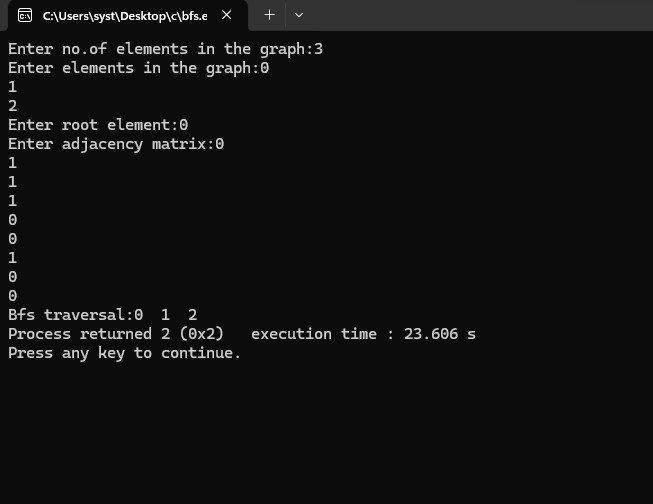
for(int j=0;j<n;j++){ scanf("%d",&a[i][j]);

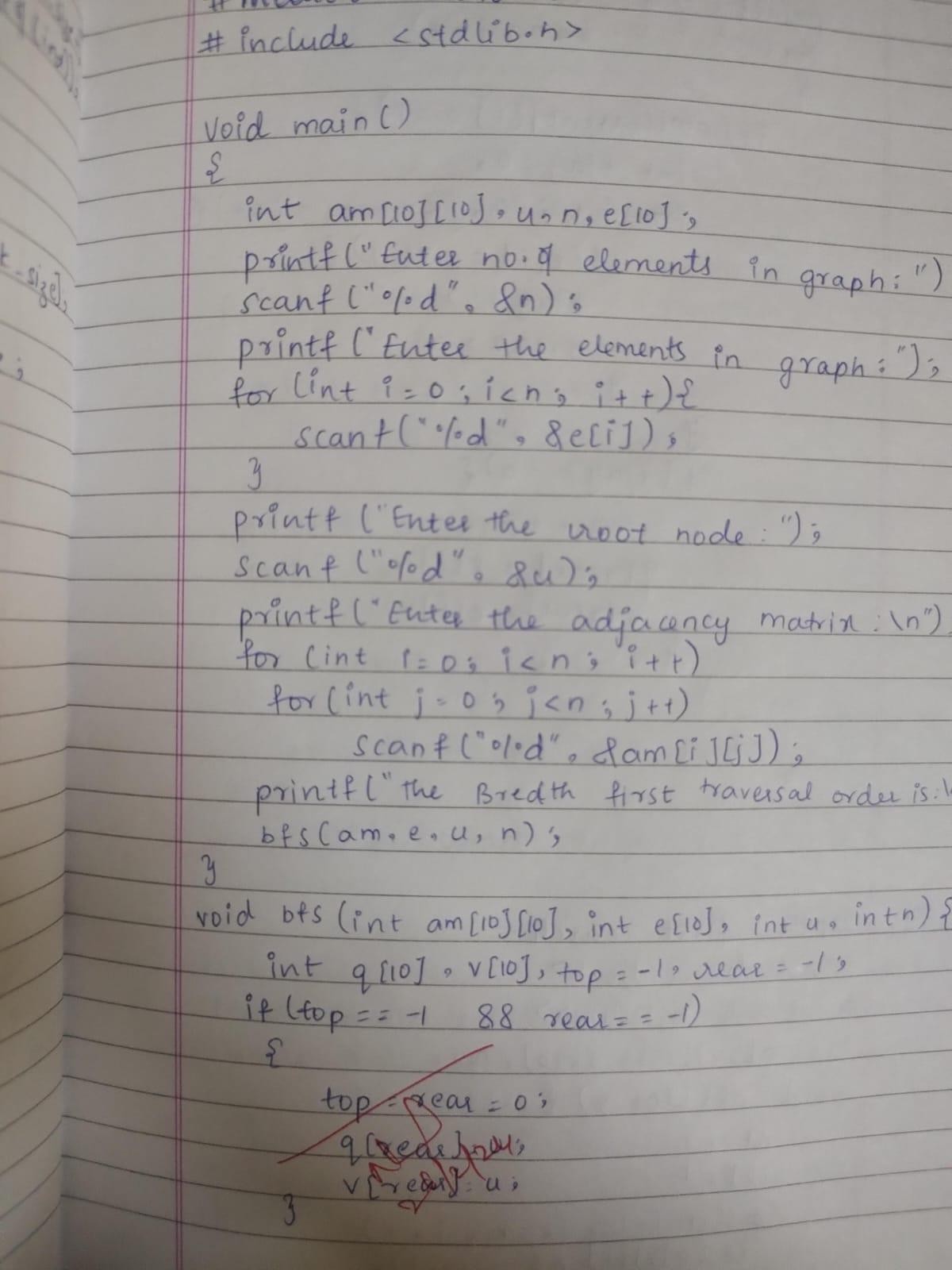
}

}

printf("Bfs traversal:"); bfs(a,g,u,n);

}





**9b) Write a program to check whether given graph is connected or not using DFS method**. #include<stdio.h>

#include<conio.h>

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

void dfs(int v)

{

int i; s[v]=1;

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

if(a[v][i] && !s[i])

{

printf("\n %d->%d",v,i); dfs(i);

}

}

int main()

{

int i, j, count=0;

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

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

{

s[i]=0;

for(j=1; j<=n; j++) a[i][j]=0;

}

printf("Enter the adjacency matrix:\n"); for(i=1; i<=n; i++)

for(j=1; j<=n; j++) scanf("%d", &a[i][j]);

dfs(1); printf("\n"); for(i=1; i<=n; i++)

{

if(s[i]) count++;

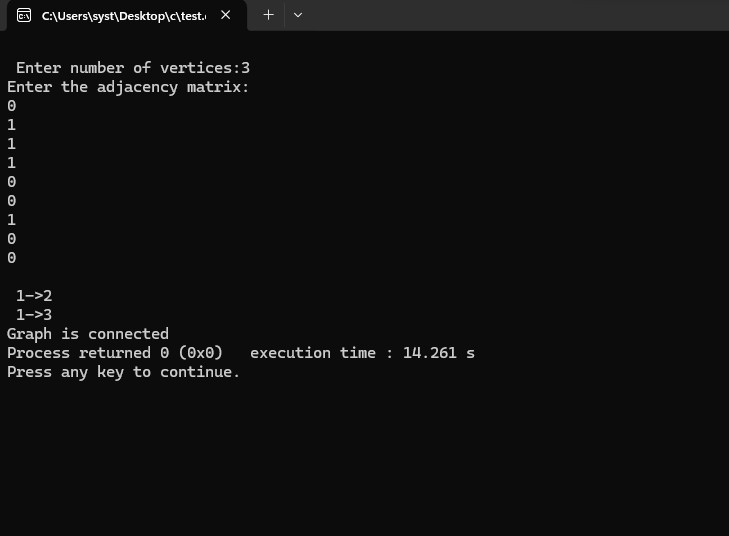
}

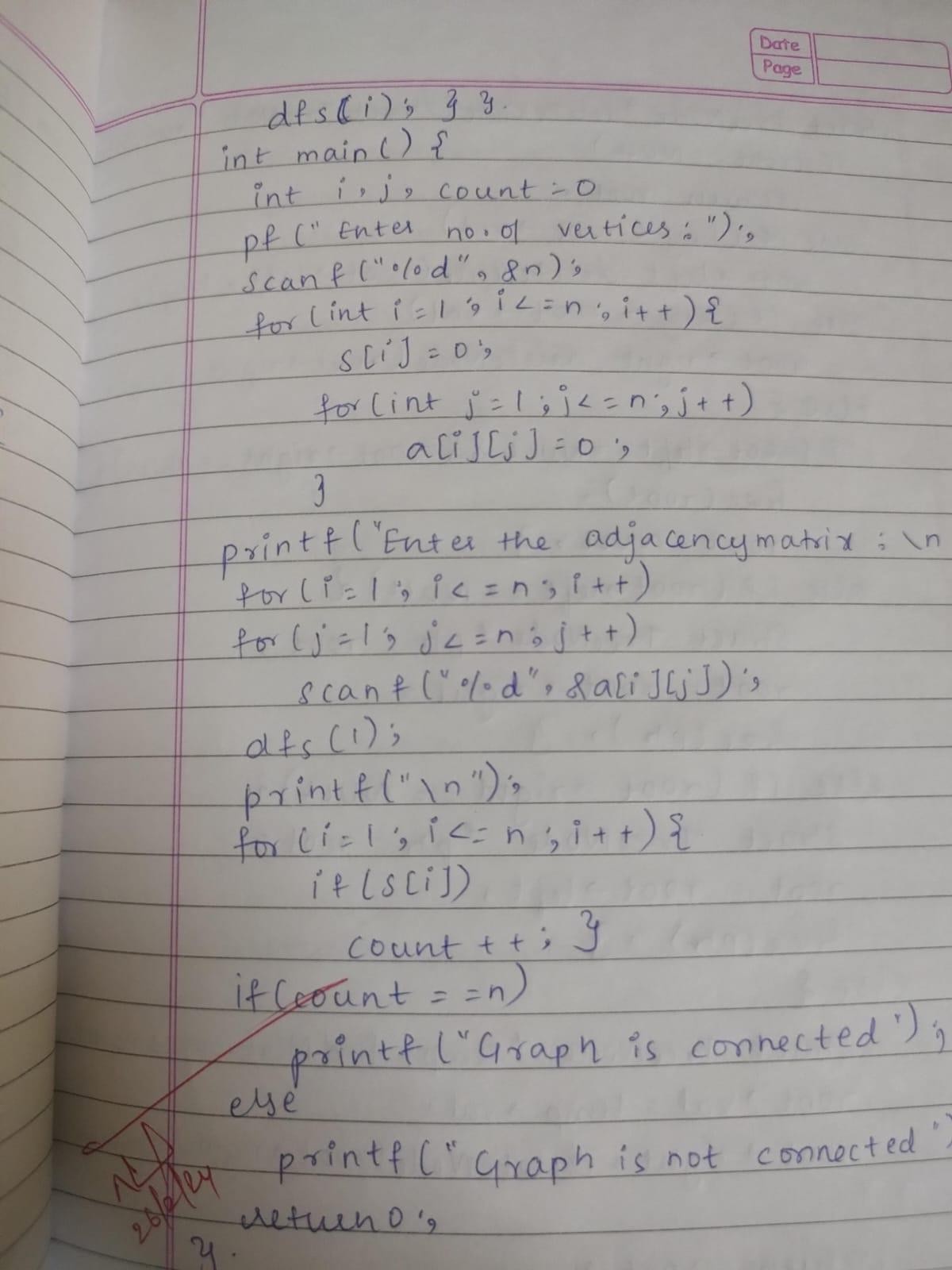
if(count==n)

printf("Graph is connected"); else

printf("Graph is not connected"); return 0;

}





**LEETCODE:**

**450.Delete a node in BST**

struct TreeNode\* findMin(struct TreeNode\* node) { while (node->left != NULL) {

node = node->left;

}

return node;

}

struct TreeNode\* deleteNode(struct TreeNode\* root, int key) { if(root==NULL)

{

return root;

}

else if(key < root->val)

{

root->left=deleteNode(root->left,key);

}

else if(key > root->val)

{

root->right=deleteNode(root->right,key);

}

else

{

if(root->left==NULL && root->right==NULL)

{

free(root); root=NULL;

}

else if(root->left==NULL)

{

struct TreeNode \*ptr=root; root=root->right; free(ptr);

}

else if(root->right==NULL)

{

struct TreeNode \*ptr=root; root=root->left; free(ptr);

}

else

{

struct TreeNode \*temp = findMin(root->right); root->val = temp->val;

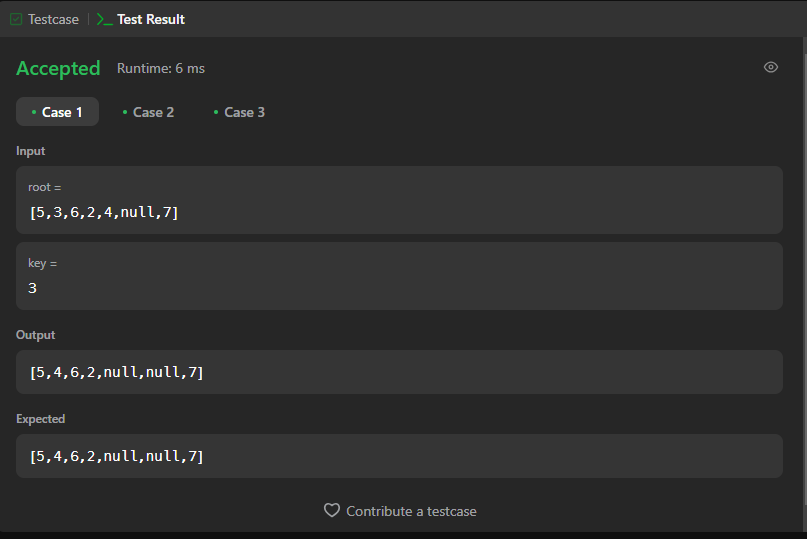
root->right = deleteNode(root->right, temp->val);

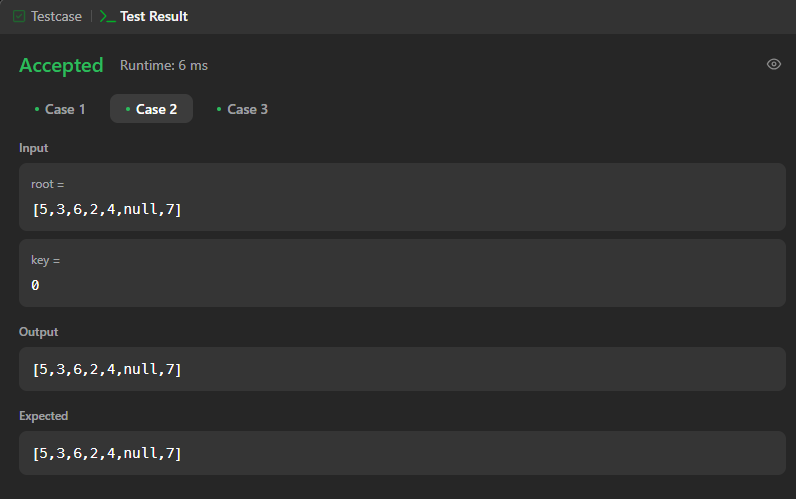
}

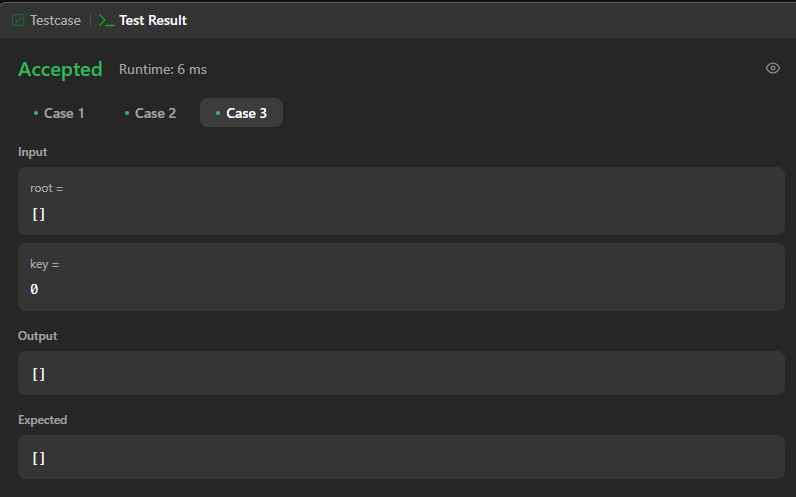
}

return root;

}







**513.Find bottom left tree value**

int findBottomLeftValue(struct TreeNode\* root) { if (root == NULL) {

return -1;

}

struct TreeNode\*\* queue = (struct TreeNode\*)malloc(pow(10,4) \* sizeof(struct TreeNode));

int front = 0, rear = 0; int leftmostValue = 0;

queue[rear++] = root; while (front < rear) {

int levelSize = rear - front;

for (int i = 0; i < levelSize; i++) {

struct TreeNode\* currentNode = queue[front++];

if (i == 0) {

leftmostValue = currentNode->val;

}

if (currentNode->left) {

queue[rear++] = currentNode->left;

}

if (currentNode->right) {

queue[rear++] = currentNode->right;

}

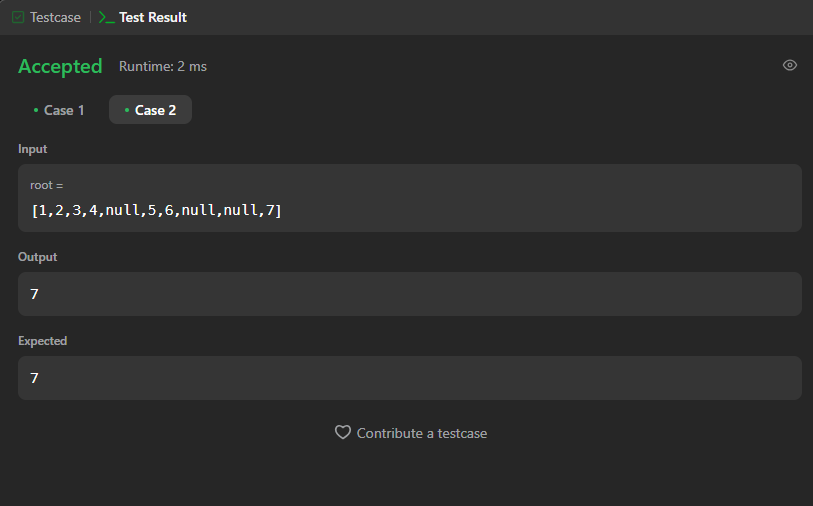
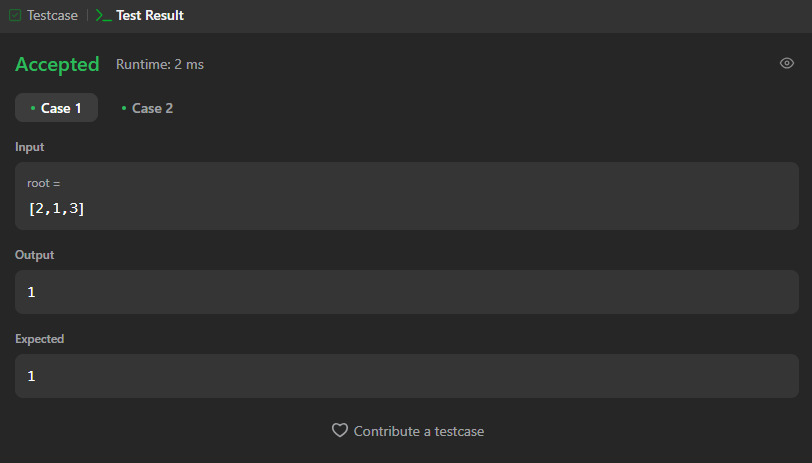
}

}

free(queue);

return leftmostValue;

}



**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> #include <stdlib.h>

#define MAX\_EMPLOYEES 1000

#define TABLE\_SIZE 100

int employeeKeys[MAX\_EMPLOYEES]; int employeeData[MAX\_EMPLOYEES]; int hashTableSize = 0;

int hash(int key) {

return key % TABLE\_SIZE;

}

void initializeHashTable() { hashTableSize = 0;

for (int i = 0; i < MAX\_EMPLOYEES; i++) { employeeKeys[i] = -1;

}

}

void insertEmployee(int key, int data) {

if (hashTableSize >= MAX\_EMPLOYEES) {

printf("Hash table is full. Cannot insert more employees.\n"); return;

}

int index = hash(key);

while (employeeKeys[index] != -1) { index = (index + 1) % TABLE\_SIZE;

}

employeeKeys[index] = key; employeeData[index] = data; hashTableSize++;

}

int searchEmployee(int key) { int index = hash(key);

while (employeeKeys[index] != -1) { if (employeeKeys[index] == key) {

return employeeData[index];

}

index = (index + 1) % TABLE\_SIZE;

}

return -1;

}

int main() { initializeHashTable();

insertEmployee(1234, 100);

insertEmployee(5678, 200);

int searchKey = 1234;

int foundData = searchEmployee(searchKey); if (foundData != -1) {

printf("Employee with key %d found. Data: %d\n", searchKey, foundData);

} else {

printf("Employee with key %d not found.\n", searchKey);

}

return 0;

}

