VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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



**LAB REPORT**

**On**

**DATA STRUCTURES (23CS3PCDST)**

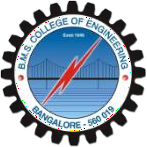
**Submitted by**

**VAISHNAVI RAMESH KULKARNI (1BM22CS316)**

**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 **VAISHNAVI RAMESH KULKARNI(1BM22CS316)**, 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--)

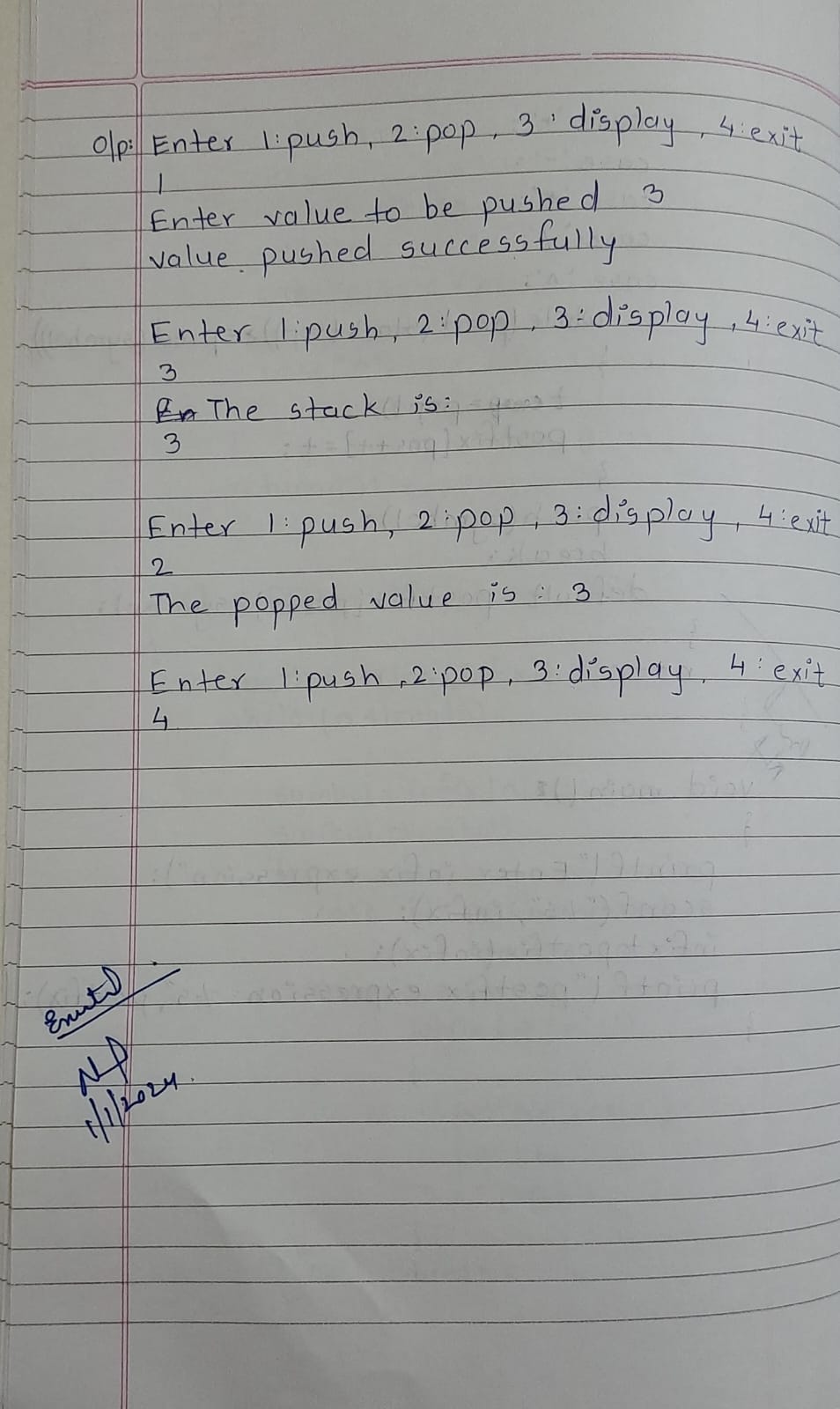
{

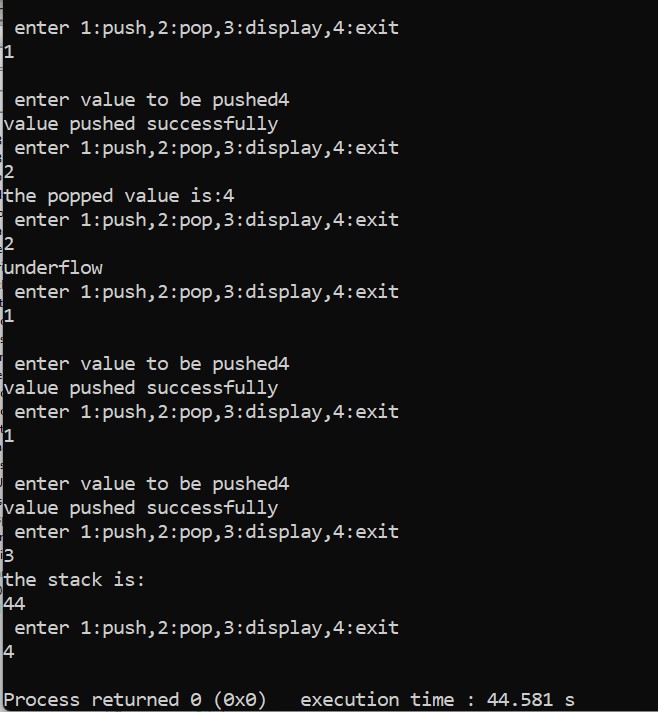
printf("%d ",stack[i]);

}

}

}





**2.WAP to convert a given valid parenthesized infix**

**arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) and /(divide)**

**#include <stdio.h>**

**#include <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)**

**{**

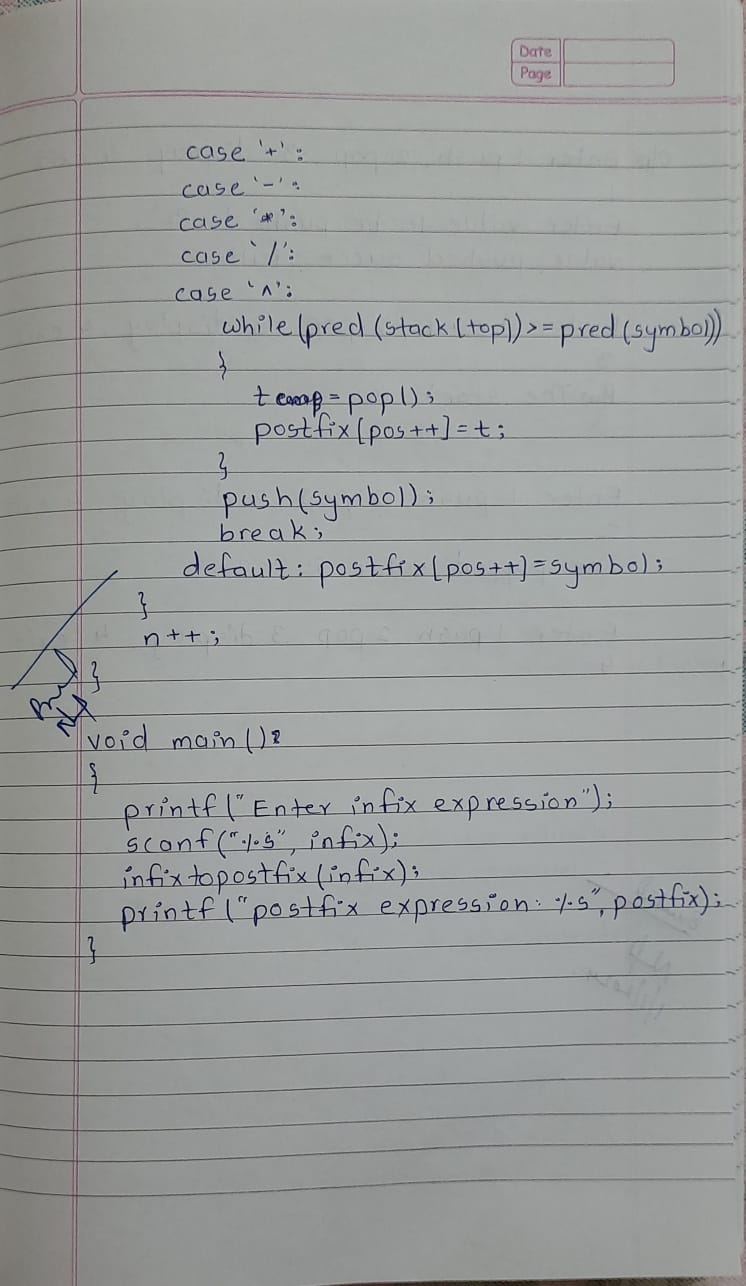
**postfix[pos] = pop();**

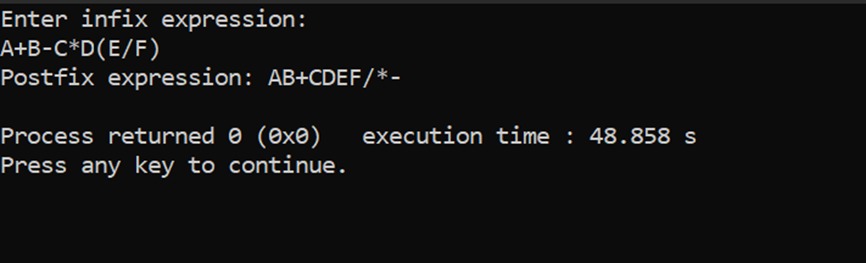
**pos++;**

**}**

**postfix[pos] = '\0';**

**}**





**3.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;

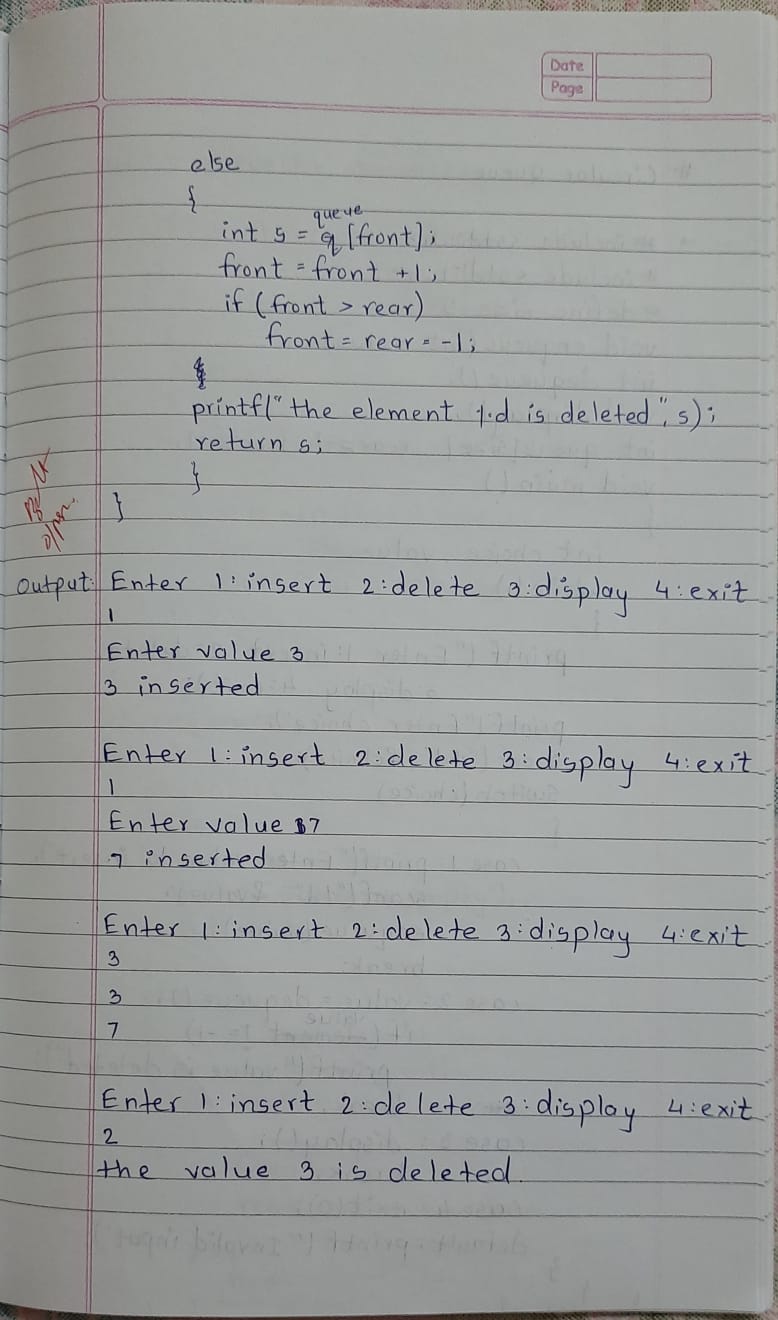
}

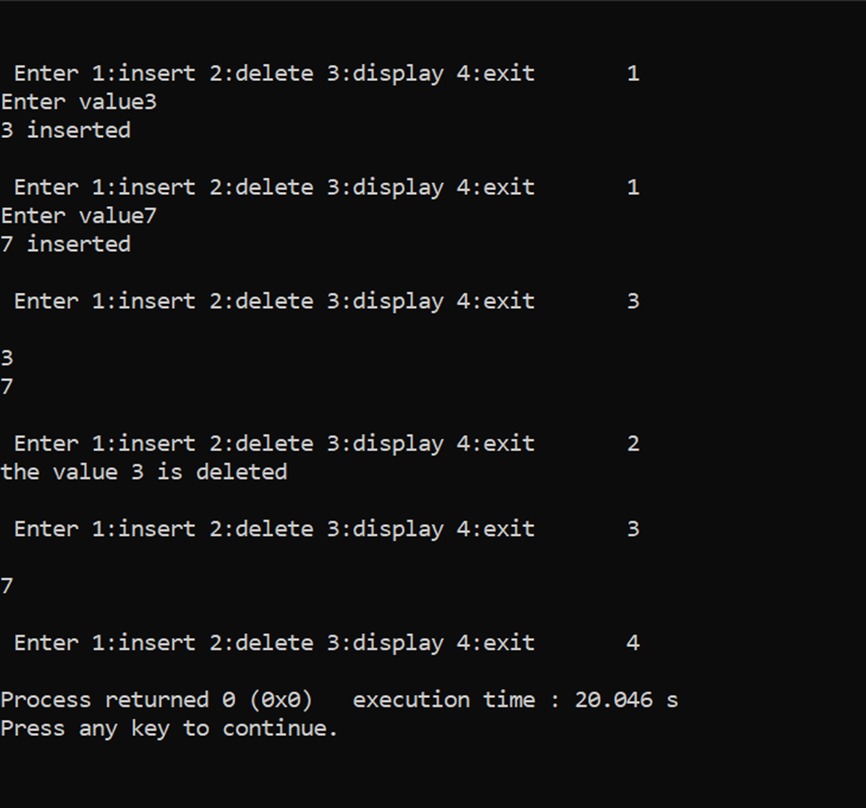
printf("\nthe element %d is deleted\n", s);

return s;

}

}





**4.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) {

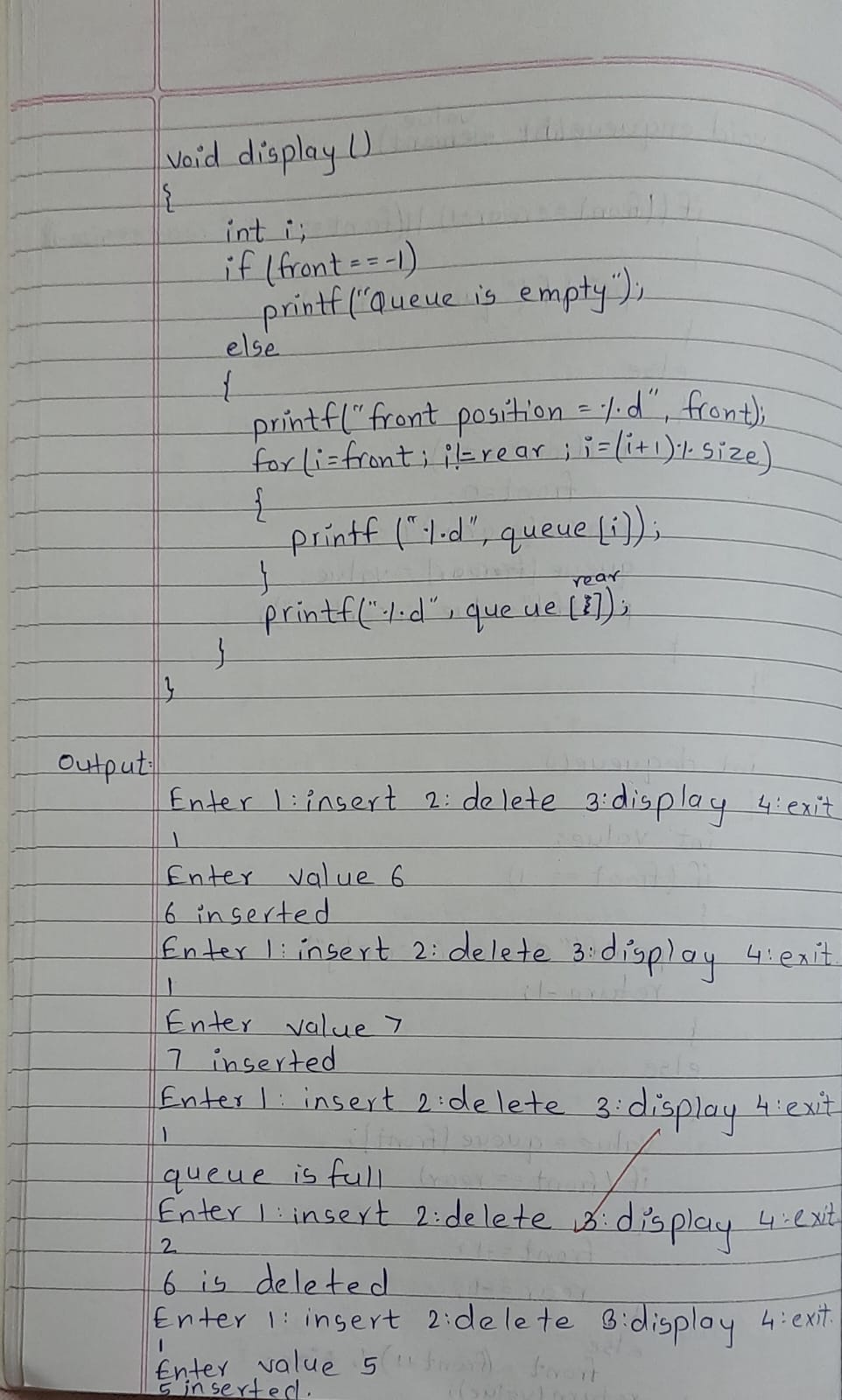
printf("%d ", q[i]);

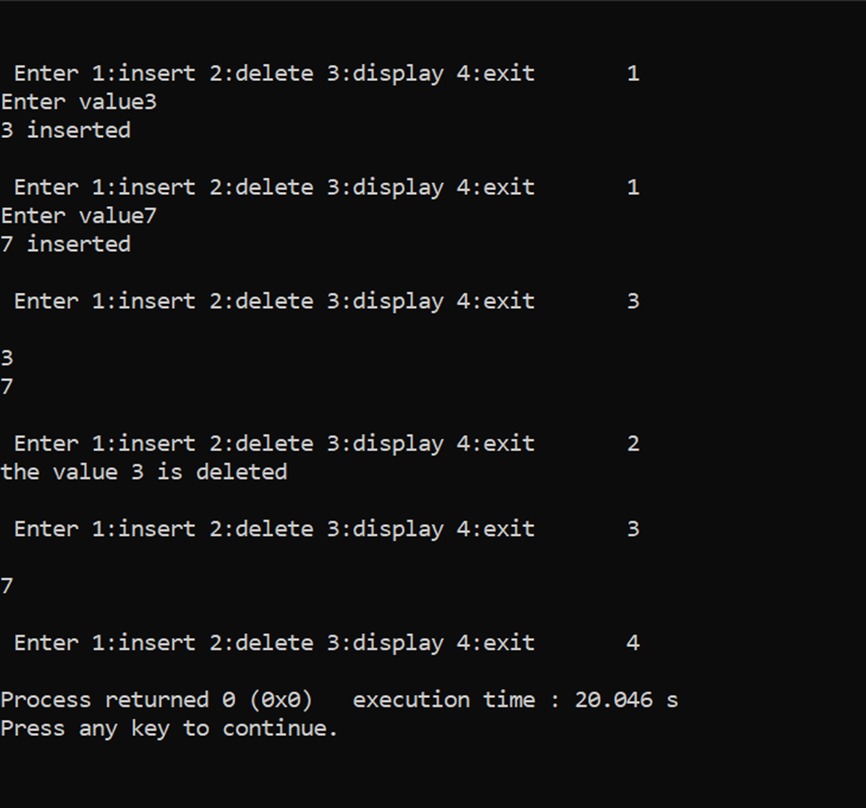
}

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);

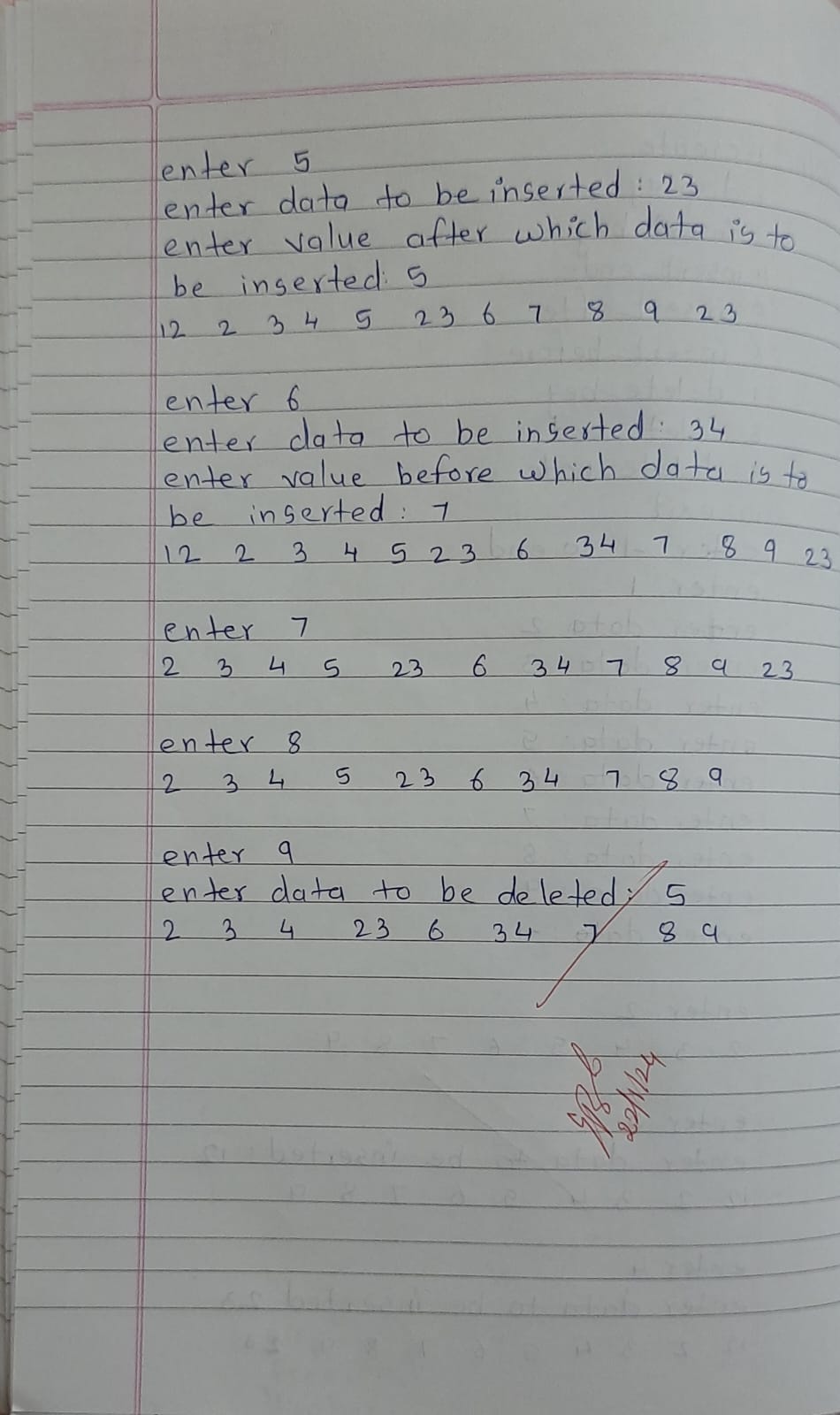
printf("\nNode with value %d deleted\n", value);

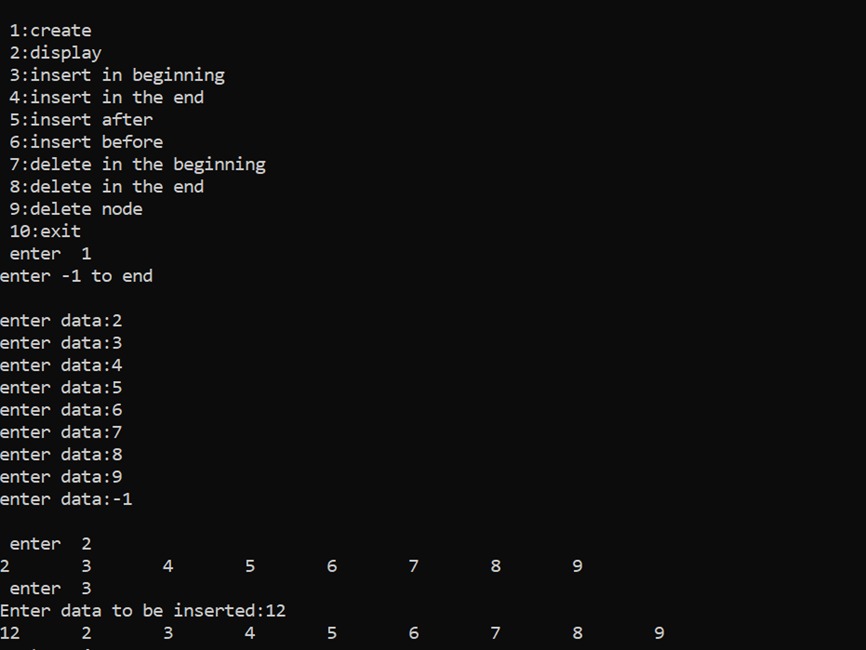
}

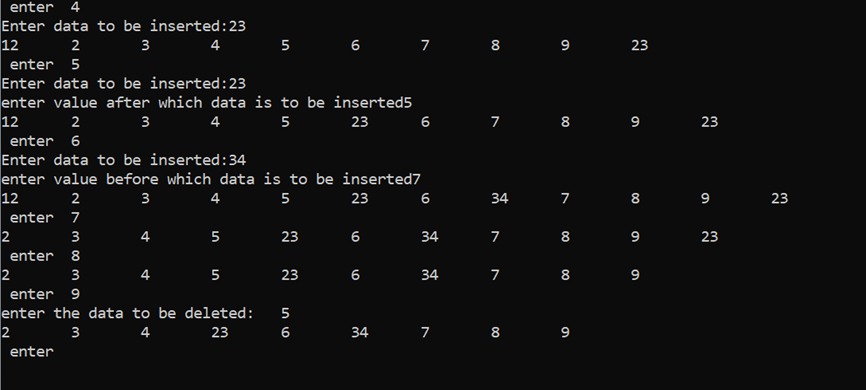
void exit\_program()

{

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







**7.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 \*head=NULL;

void create()

{

struct node \*new\_node,\*ptr;

int num;

printf("\n enter -1 to end\n");

printf("enter data: \n");

scanf("%d",&num);

while(num!=-1)

{

new\_node=(struct node\*)malloc(sizeof(struct node));

new\_node->data=num;

if(head==NULL)

{

new\_node->next=NULL;

head=new\_node;

}

else

{

ptr=head;

while(ptr->next!=NULL)

ptr=ptr->next;

new\_node->next=NULL;

ptr->next=new\_node;

}

printf("enter data:");

scanf("%d",&num);

}

}

void display()

{

struct node \*ptr;

ptr=head;

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

void sort()

{

struct node \*ptr1,\*ptr2;

int temp;

ptr1=head;

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 con()

{

struct node \*new\_node,\*h1,\*h2,\*ptr;

int i,n,m;

printf("enter no. of elements in 1st list:\n");

scanf("%d",&n);

h1=NULL;

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

{

printf("enter data:\n");

new\_node=malloc(sizeof(struct node));

scanf("%d",&new\_node->data);

new\_node->next=NULL;

if(h1==NULL)

{

h1=new\_node;

}

else

{

ptr=h1;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

}

}

printf("enter no. of elements in 2nd list:\n");

scanf("%d",&m);

h2=NULL;

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

{

printf("enter data:\n");

new\_node=malloc(sizeof(struct node));

scanf("%d",&new\_node->data);

new\_node->next=NULL;

if(h2==NULL)

{

h2=new\_node;

}

else

{

ptr=h2;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

}

}

concatenate(h1,h2);

}

void concatenate(struct node \*h1,struct node \*h2)

{

struct node \*ptr;

head=h1;

ptr=head;

if(h1==NULL && h2==NULL)

{

printf("list is empty!");

}

else

{

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=h2;

display();

}

}

void reverse()

{

struct node \*current , \*ptr , \*temp;

ptr=NULL;

current=head;

while(current!=NULL)

{

temp=current->next;

current->next=ptr;

ptr=current;

current=temp;

}

head=ptr;

display();

}

void main()

{

int choice;

while(1)

{

printf("\n Enter \n 1:create \n 2:sort \n 3:concatenate \n 4:reverse \n 5:exit");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

sort();

display();

break;

case 3:

con();

break;

case 4:

reverse();

break;

case 5:

exit(0);

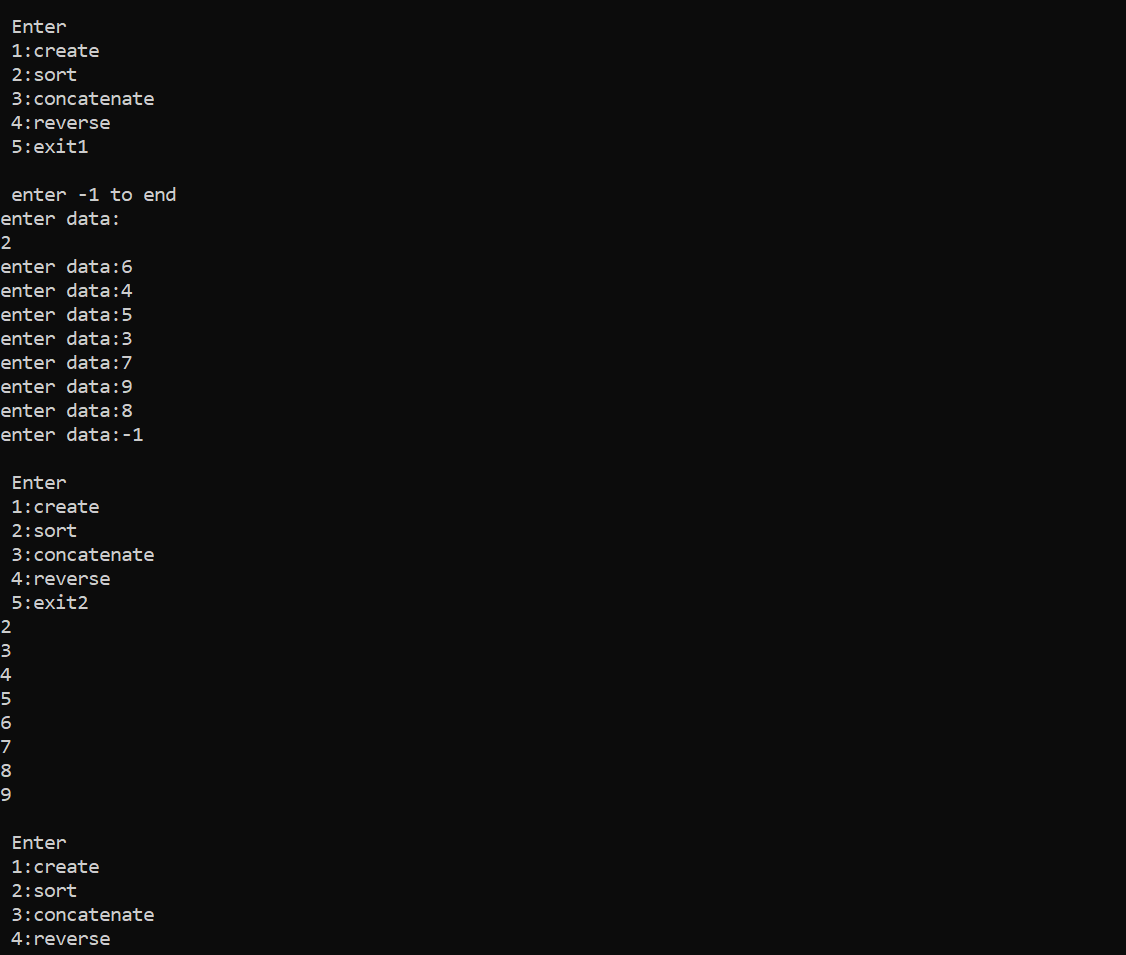
default:

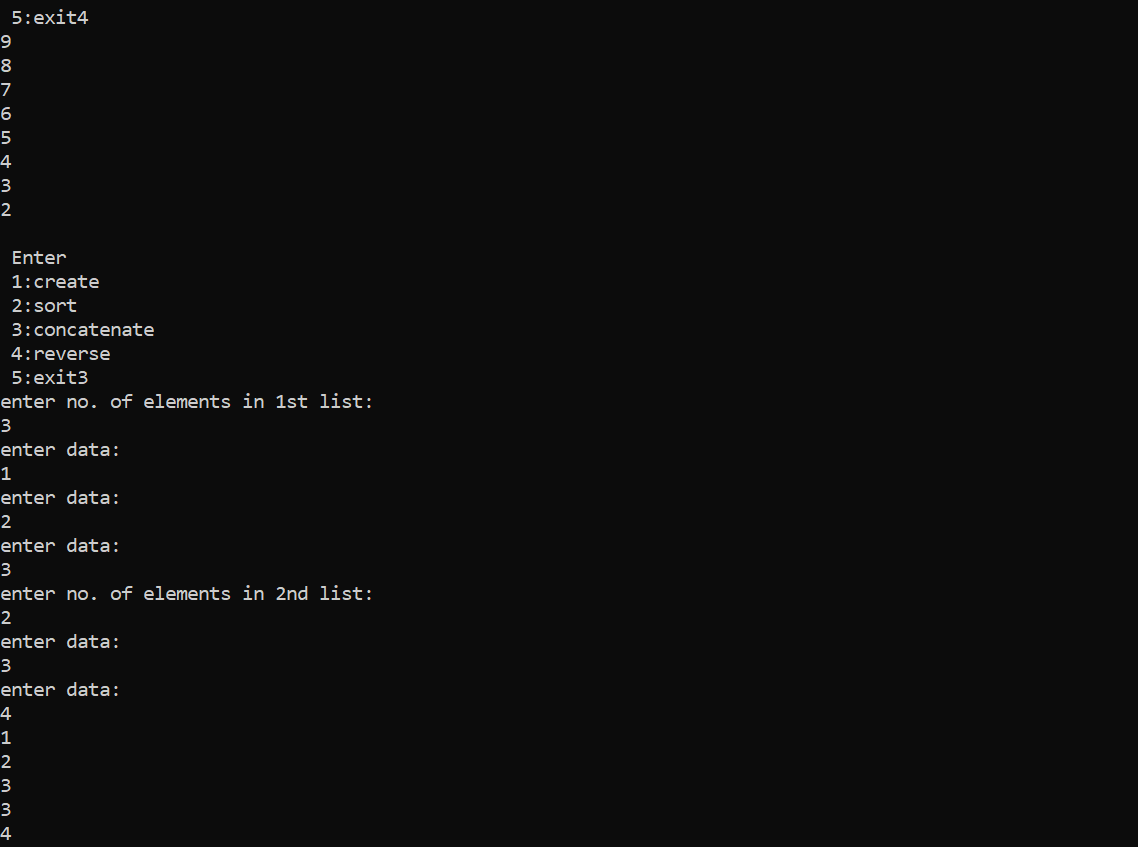
printf("invalid input");

}

}

}





**Stack Implementation using single linked list:**

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int data;

struct node \*next;

};

struct node\* top=NULL;

void main()

{

int choice;

while(1)

{

printf("\n enter \n 1:push \n 2:pop \n 3:display 4:exit");

scanf("%d",&choice);

switch(choice)

{

case 1:

push();

printf("\n the stack is:\n");

display();

break;

case 2:

pop();

printf("\n the stack is:\n");

display();

break;

case 3:

printf("\n the stack is:\n");

display();

break;

case 4:

exit(0);

default:

printf("Invalid Input!");

}

}

}

void push()

{

struct node \*temp;

int val;

printf("enter the data to be pushed:");

scanf("%d",&val);

if(top==NULL)

{

top=malloc(sizeof(struct node));

top->data=val;

top->next=NULL;

}

else

{

temp=malloc(sizeof(struct node));

temp->data=val;

temp->next=top;

top=temp;

}

}

void pop()

{

struct node \*temp;

if(top==NULL)

{

printf("Stack is Empty!");

}

else

{

temp=top;

top=top->next;

printf("\n the data popped is %d ",temp->data);

free(temp);

}

}

void display()

{

struct node \*temp;

temp=top;

if(top==NULL)

{

printf("stack is empty!");

}

else

{

while(temp!=NULL)

{

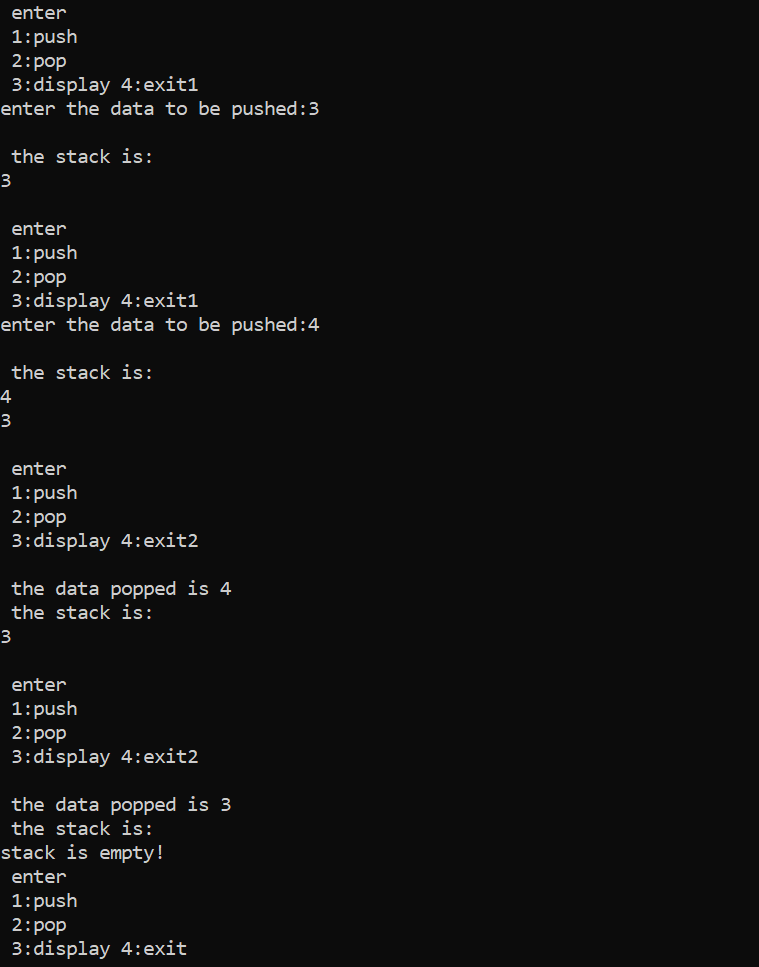
printf("%d\n",temp->data);

temp=temp->next;

}

}

}



**Queue Implementation using linked list:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

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

void enqueue();

void dequeue();

void display();

int main()

{

int choice;

while(1)

{

printf("\n Enter \n 1:enqueue \n 2:dequeue \n 3:display \n 4:exit \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

enqueue();

printf("the queue is:");

display();

break;

case 2:

dequeue();

printf("\n the queue is:");

display();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Invalid Input!");

}

}

return 0;

}

void enqueue()

{

int val;

printf("enter data to be inserted:\n");

scanf("%d",&val);

struct node \*temp;

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

temp->data=val;

temp->next=NULL;

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

{

front=rear=temp;

}

else

{

rear->next=temp;

rear=temp;

}

}

void dequeue()

{

struct node \*temp;

temp=front;

front=front->next;

printf("the removed data is: %d ",temp->data);

free(temp);

}

void display()

{

struct node \*temp;

temp=front;

if(front==NULL)

{

printf("Queue is empty!");

}

else

{

while(temp!=NULL)

{

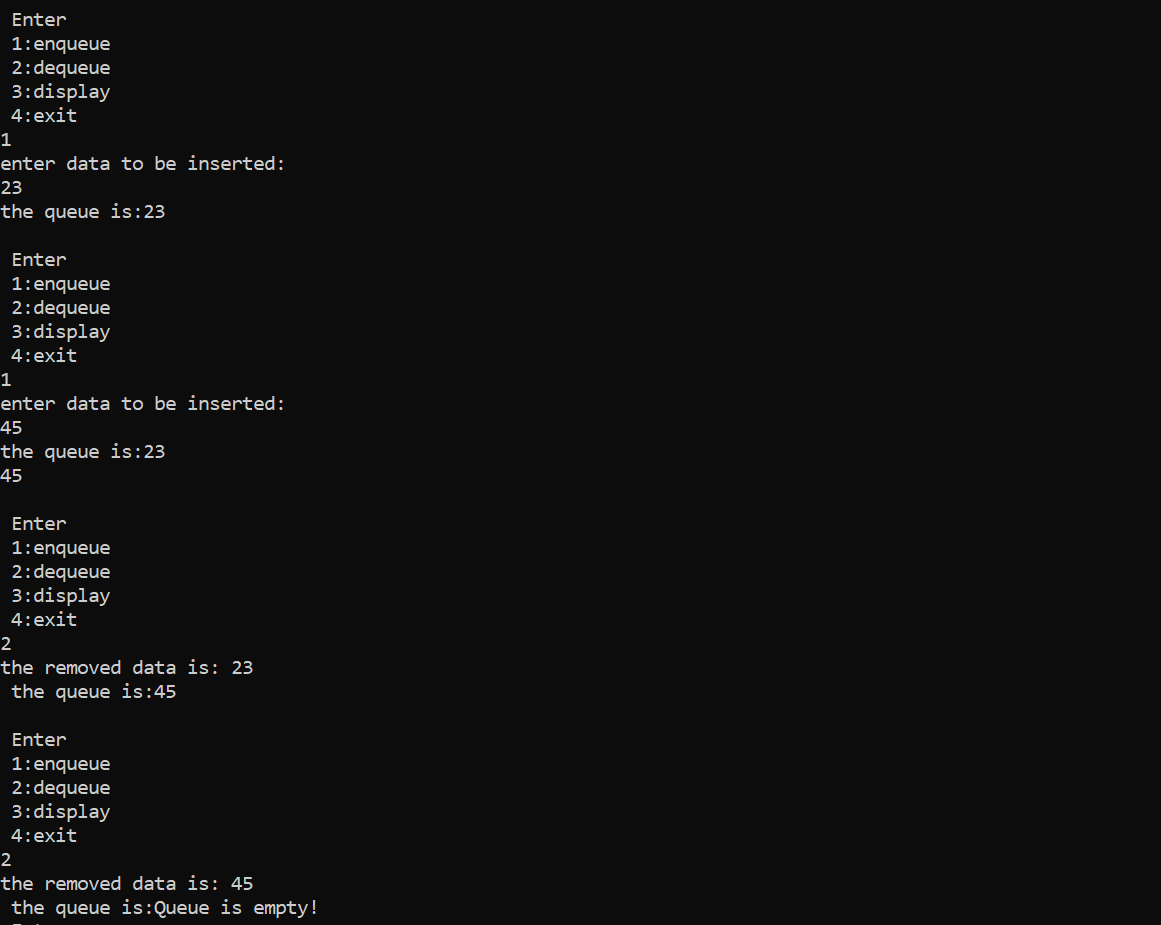
printf("%d \n",temp->data);

temp=temp->next;

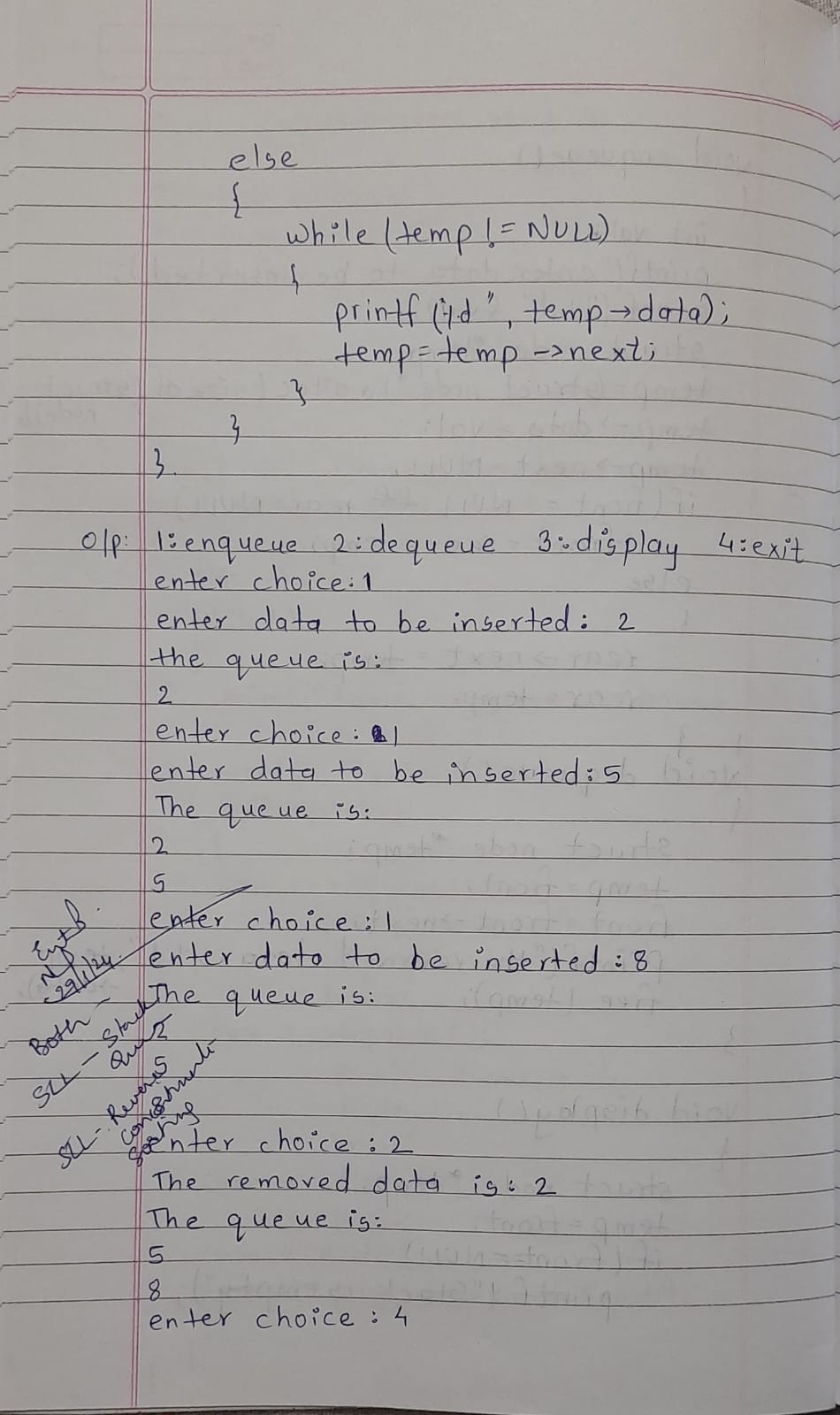
}

}

}



**Observation book:**



**9. 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**

#include <stdio.h>

#include <stdlib.h>

struct node

{

struct node \*prev;

int data;

struct node \*next;

}\*head=NULL;

void display();

void create();

void insert\_left();

void delete\_node();

struct node \*new\_node,\*ptr;

void main()

{

int choice;

printf("1:create \n 2:insert\_left \n 3:delete\_node\n 4:exit");

while(1)

{

printf("\nenter choice: \t");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

insert\_left();

break;

case 3:

delete\_node();

break;

case 4:

exit(0);

default:

printf("invalid input");

}

}

}

void create()

{

int val;

printf("\nenter -1 to end");

printf("\nenter data:\t");

scanf("%d",&val);

while(val!=-1)

{

new\_node=malloc(sizeof(struct node));

new\_node->data=val;

if(head==NULL)

{

new\_node->prev=NULL;

new\_node->next=NULL;

head=new\_node;

}

else

{

ptr=head;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=new\_node;

new\_node->prev=ptr;

new\_node->next = NULL;

}

printf("\n enter data:\t");

scanf("%d",&val);

}

display();

}

void display()

{

if(head==NULL)

printf("the list is empty");

else

{

ptr=head;

printf("\n The list is:\n");

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

}

void insert\_left()

{

int val,d;

printf("enter value to be inserted:");

scanf("%d",&val);

printf("enter value before which data is to be inserted:");

scanf("%d",&d);

new\_node=malloc(sizeof(struct node));

new\_node->data=val;

if(head==NULL)

{

new\_node->prev=NULL;

new\_node->next=NULL;

head=new\_node;

}

else

{

ptr=head;

while(ptr->data!=d)

ptr=ptr->next;

if(ptr==head)

{

new\_node->prev=NULL;

new\_node->next=head;

head->prev=new\_node;

head=new\_node;

}

else

{

new\_node->next=ptr;

new\_node->prev=ptr->prev;

ptr->prev->next=new\_node;

ptr->prev=new\_node;

}

}

display();

}

void delete\_node()

{

if(head==NULL)

printf("\n the list is empty");

else

{

int val;

printf("enter value to be deleted:\n");

scanf("%d",&val);

ptr=head;

while(ptr->data!=val)

ptr=ptr->next;

if(ptr==head)

{

head=head->next;

head->prev=NULL;

free(ptr);

}

else

{

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

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

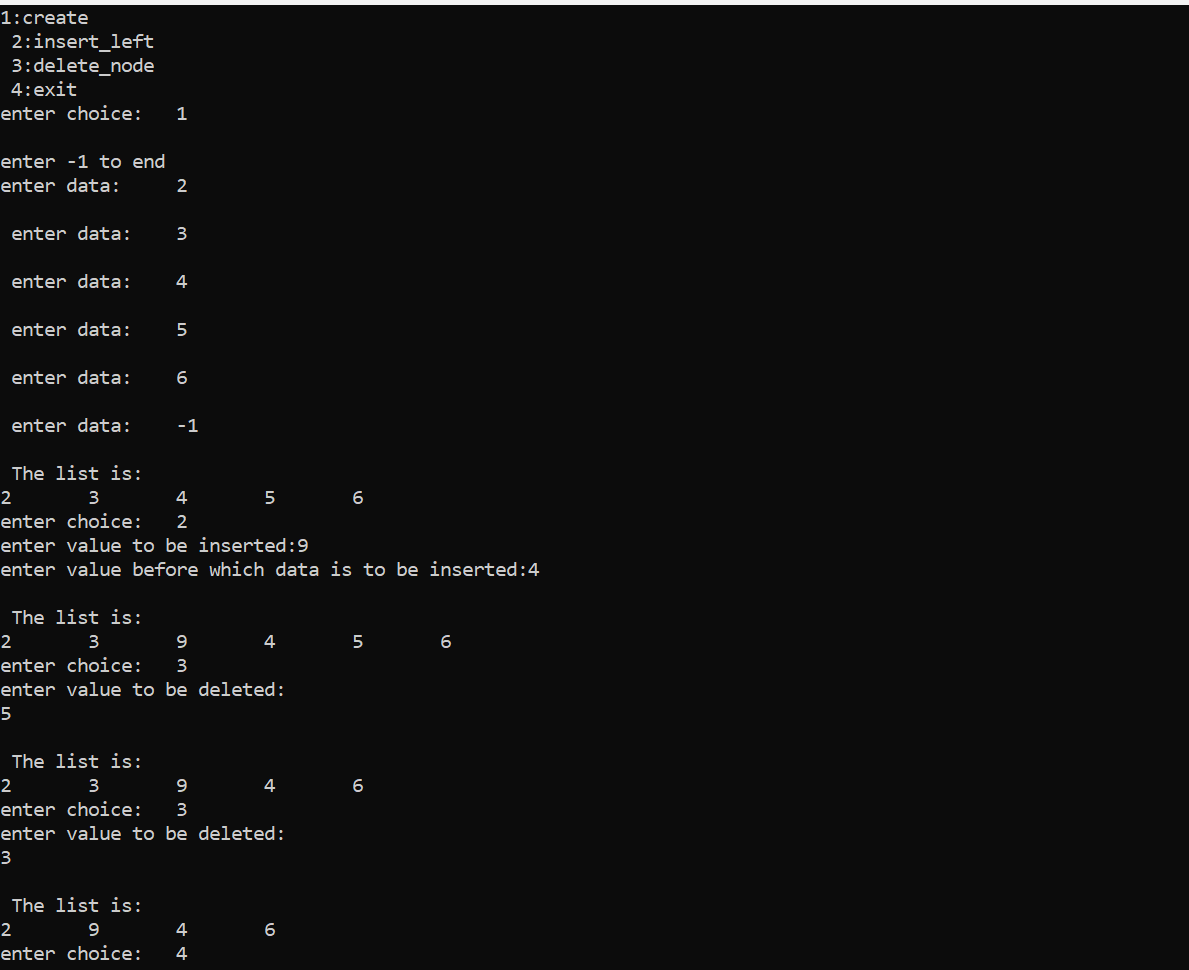
free(ptr);

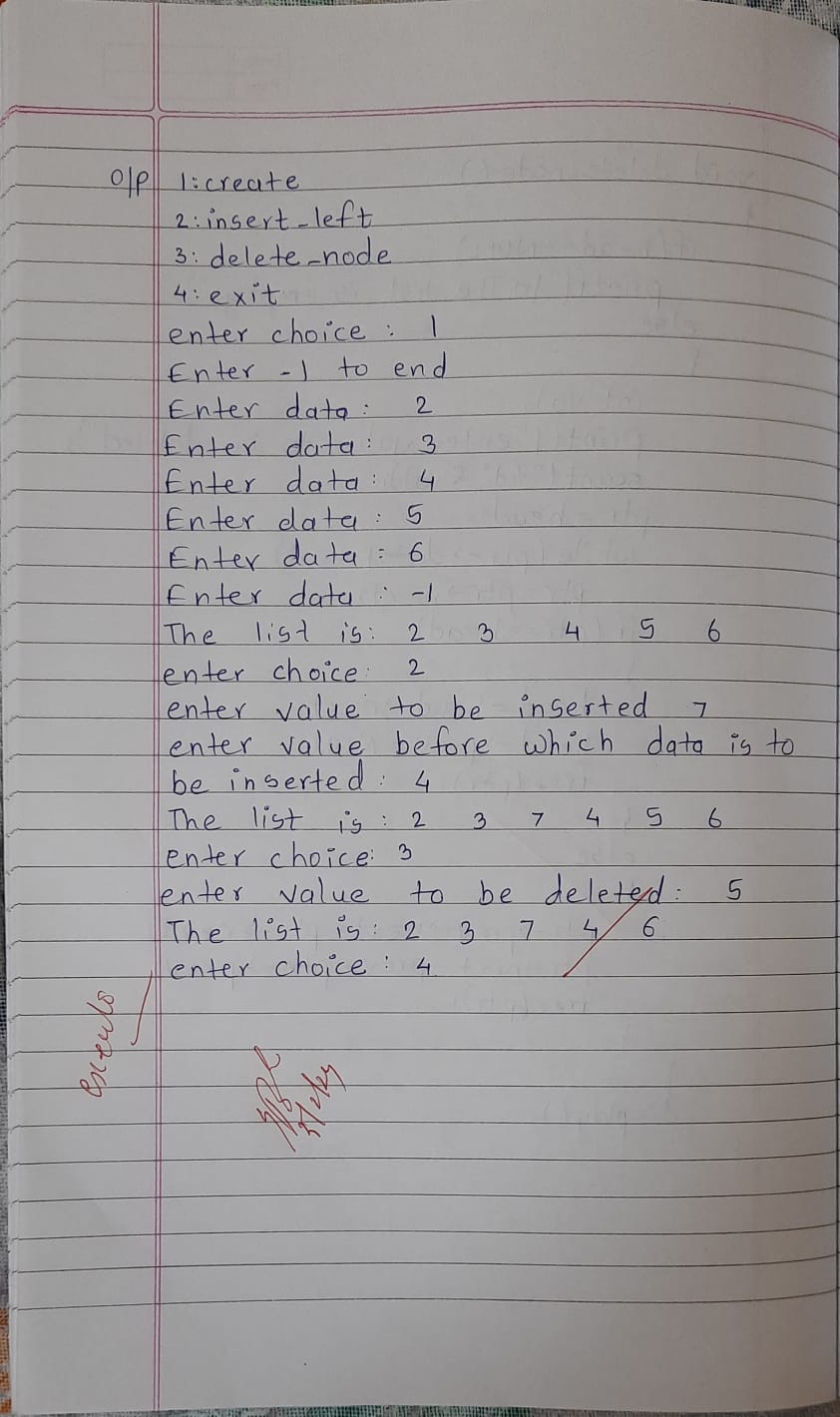
}

}

display();

}





LEETCODE PROGRAM: SCORE OF PARENTHESES

int scoreOfParentheses(char\* s) {

    if (s == NULL || \*s == '\0')

        return 0;

    int stack[50];

    int top = 0;

    int score = 0;

    for (int i = 0; s[i] != '\0'; i++) {

        if (s[i] == '(') {

            if (top >= 50) {

                return 0;

            }

            stack[top++] = score;

            score = 0;

        } else {

            int previousScore = stack[--top];

            if (score > 0) {

                score = previousScore + 2 \* score;

            } else {

                score = previousScore + 1;

            }

        }

    }

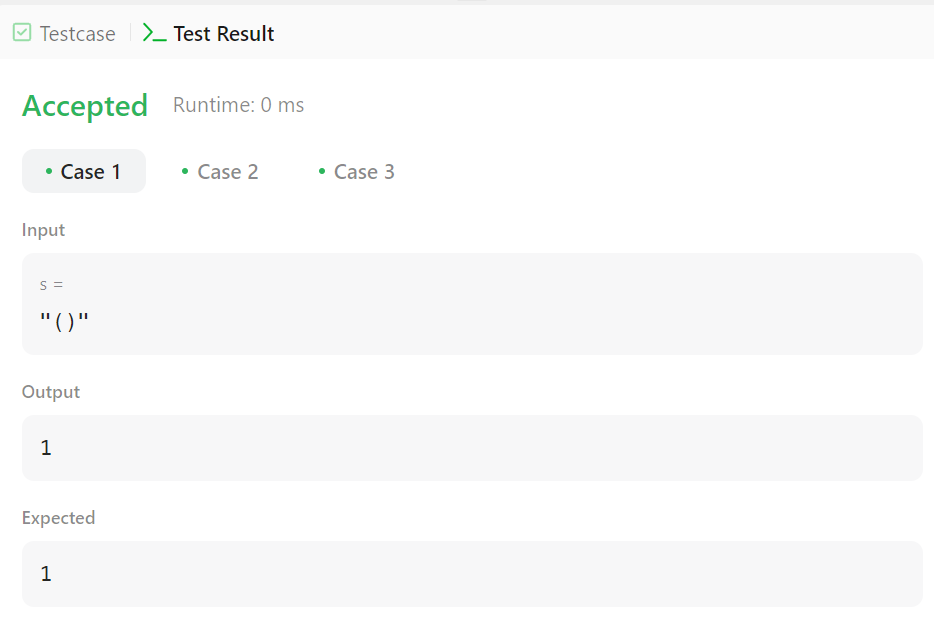
    if (top != 0) {

        return 0;

    }

    return score;

}



LEETCODE PROGRAM:ODD EVEN LINKED LIST

struct ListNode\* oddEvenList(struct ListNode\* head) {

    struct ListNode \*odd = head;

    struct ListNode \*even = head->next;

    struct ListNode \*even\_head = even;

    while (even != NULL && even->next != NULL) {

        odd->next = even->next;

        odd = odd->next;

        even->next = odd->next;

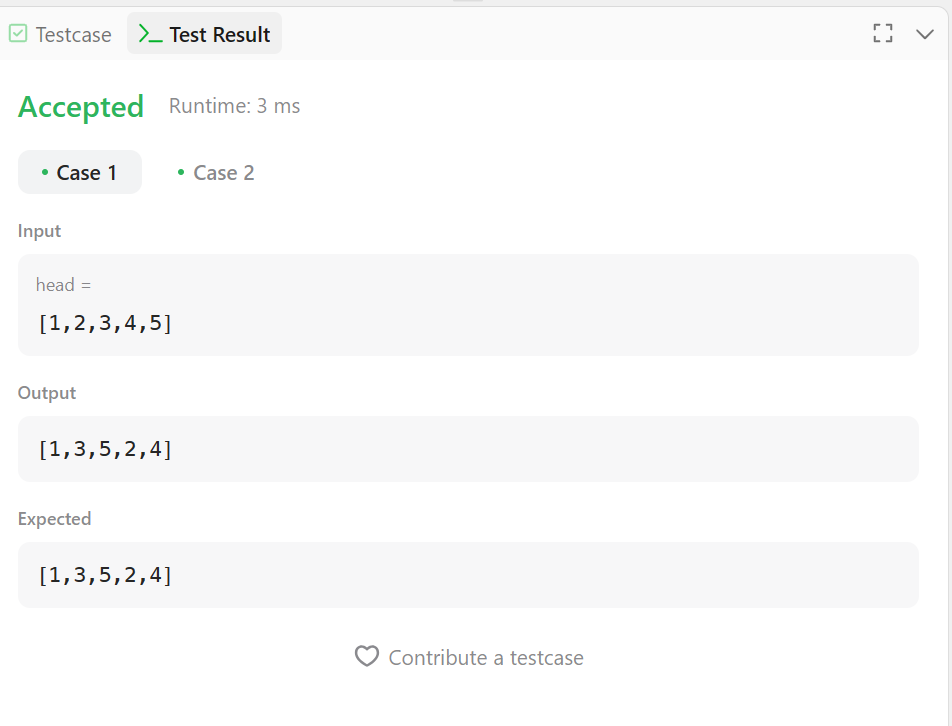
        even = even->next;

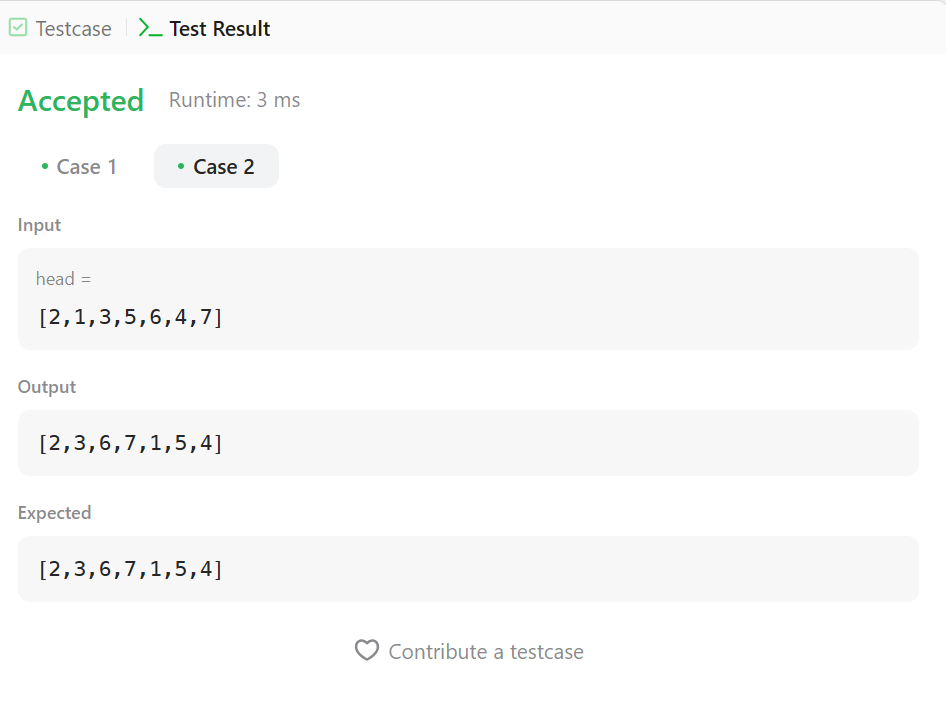
    }

    odd->next = even\_head;

    return head;

}





LEETCODE PROGRAM: DELETE THE MIDDLE NODE OF A LINKED LIST

struct ListNode\* deleteMiddle(struct ListNode\* head) {

    if (head == NULL || head->next == NULL) {

        return head;

    }

    struct ListNode\* ptr = head;

    struct ListNode\* preptr = NULL;

    int count = 0;

    while (ptr != NULL) {

        count++;

        ptr = ptr->next;

    }

    ptr = head;

    for (int i = 0; i < count / 2; i++) {

        preptr = ptr;

        ptr = ptr->next;

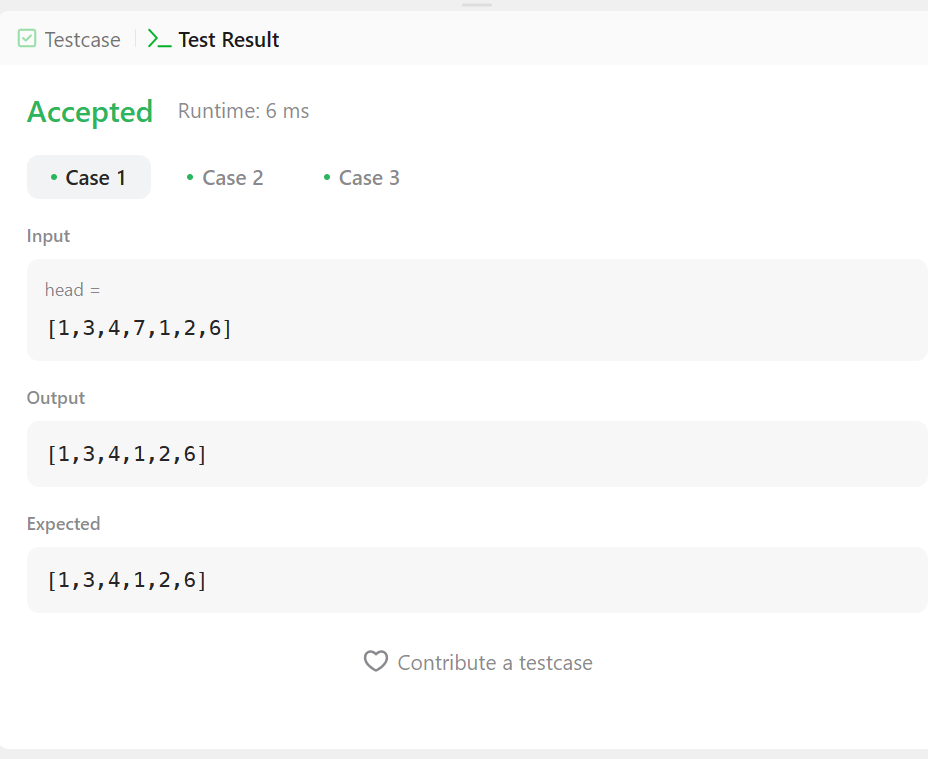
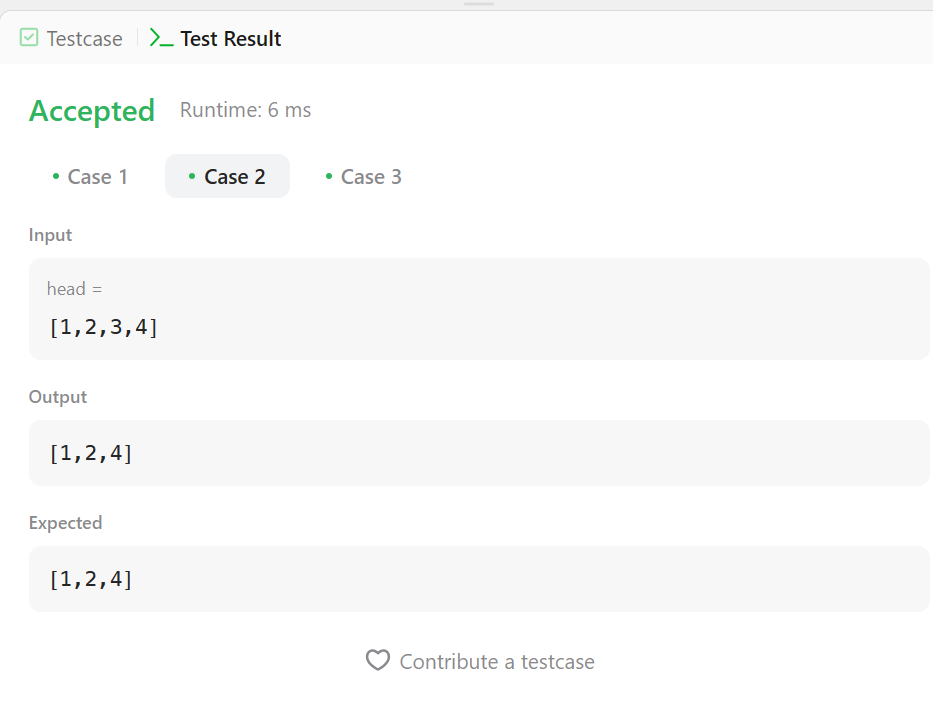
    }

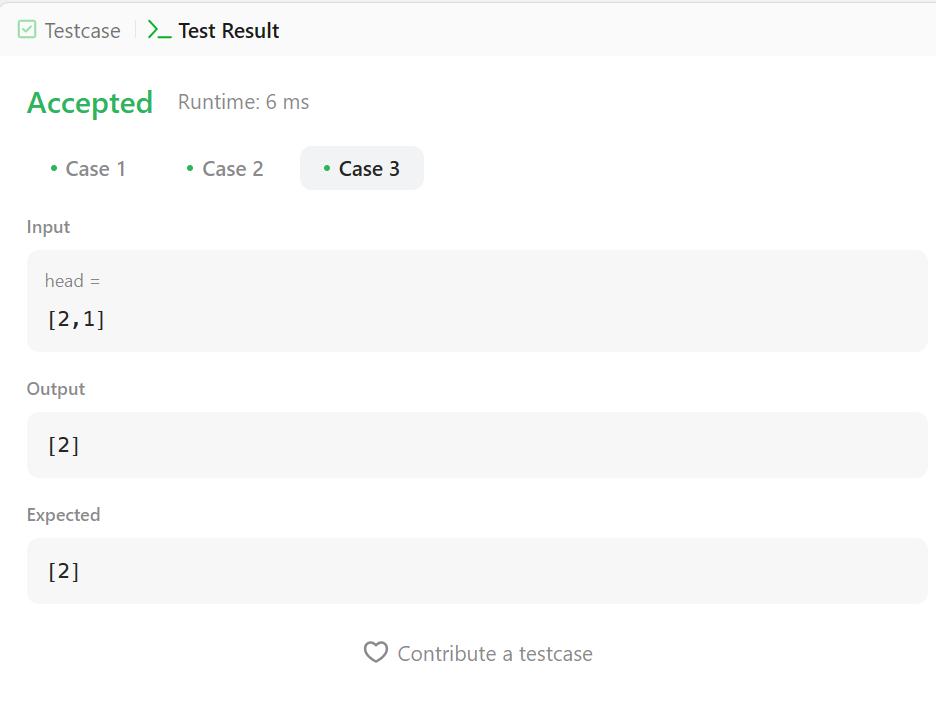
    preptr->next = ptr->next;

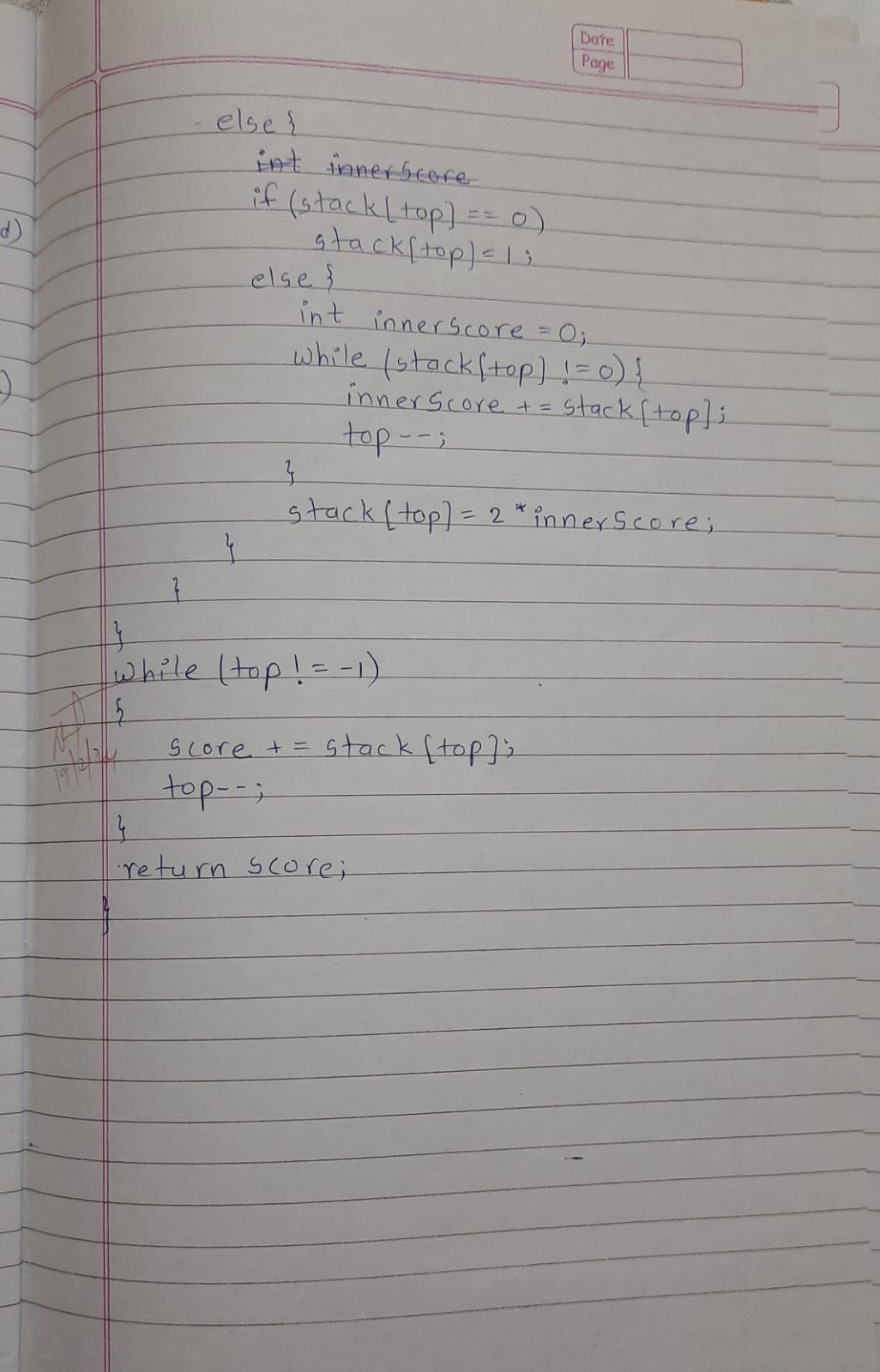
    free(ptr);

    return head;

}





**10.Write a program.**

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

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left,\*right;

}\*root;

struct node \*create()

{

struct node \*temp;

int val;

printf("enter data:");

scanf("%d",&val);

temp = malloc(sizeof(struct node));

temp->data = val;

temp->left=temp->right=NULL;

return temp;

};

void insert(struct node \*root,struct node \*temp)

{

if(temp->data<root->data)

{

if(root->left!=NULL)

insert(root->left,temp);

else

root->left=temp;

}

if(temp->data>root->data)

{

if(root->right!=NULL)

insert(root->right,temp);

else

root->right=temp;

}

}

void inorder(struct node \*root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d \t",root->data);

inorder(root->right);

}

}

void preorder(struct node \*root)

{

if(root!=NULL)

{

printf("%d \t",root->data);

preorder(root->left);

preorder(root->right);

}

}

void postorder(struct node \*root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%d \t",root->data);

}

}

void main()

{

int n,choice;

struct node \*temp;

root=NULL;

printf("enter no. of elements in tree");

scanf("%d",&n);

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

{

temp=create();

if(root==NULL)

root=temp;

else

insert(root,temp);

}

printf("\n1:inorder \n2:preorder \n3:postorder \n4:exit");

while(1)

{

printf("\nenter choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Inorder:");

inorder(root);

break;

case 2:

printf("preorder:");

preorder(root);

break;

case 3:

printf("postorder:");

postorder(root);

case 4:

exit(0);

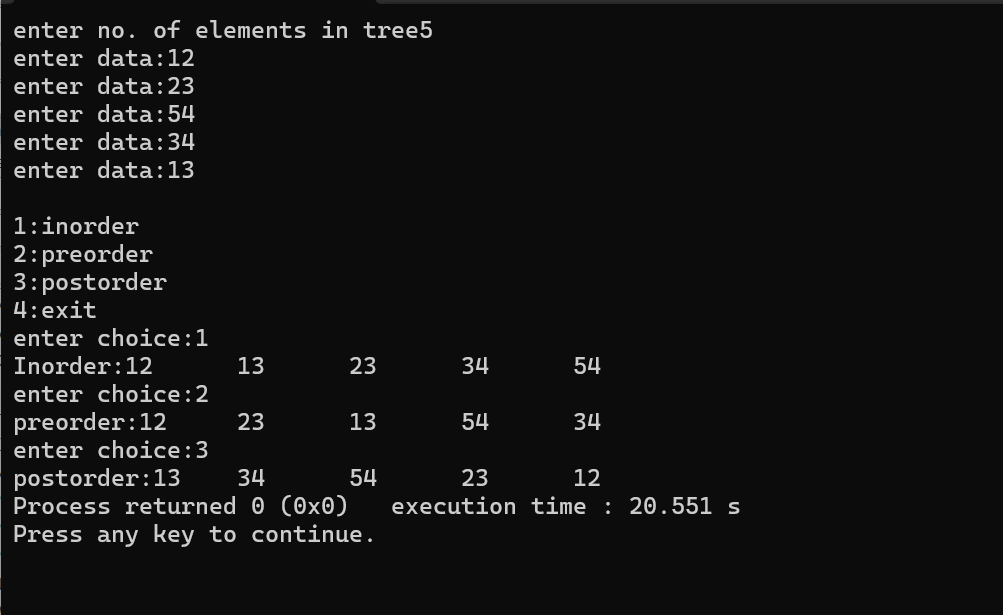
default:

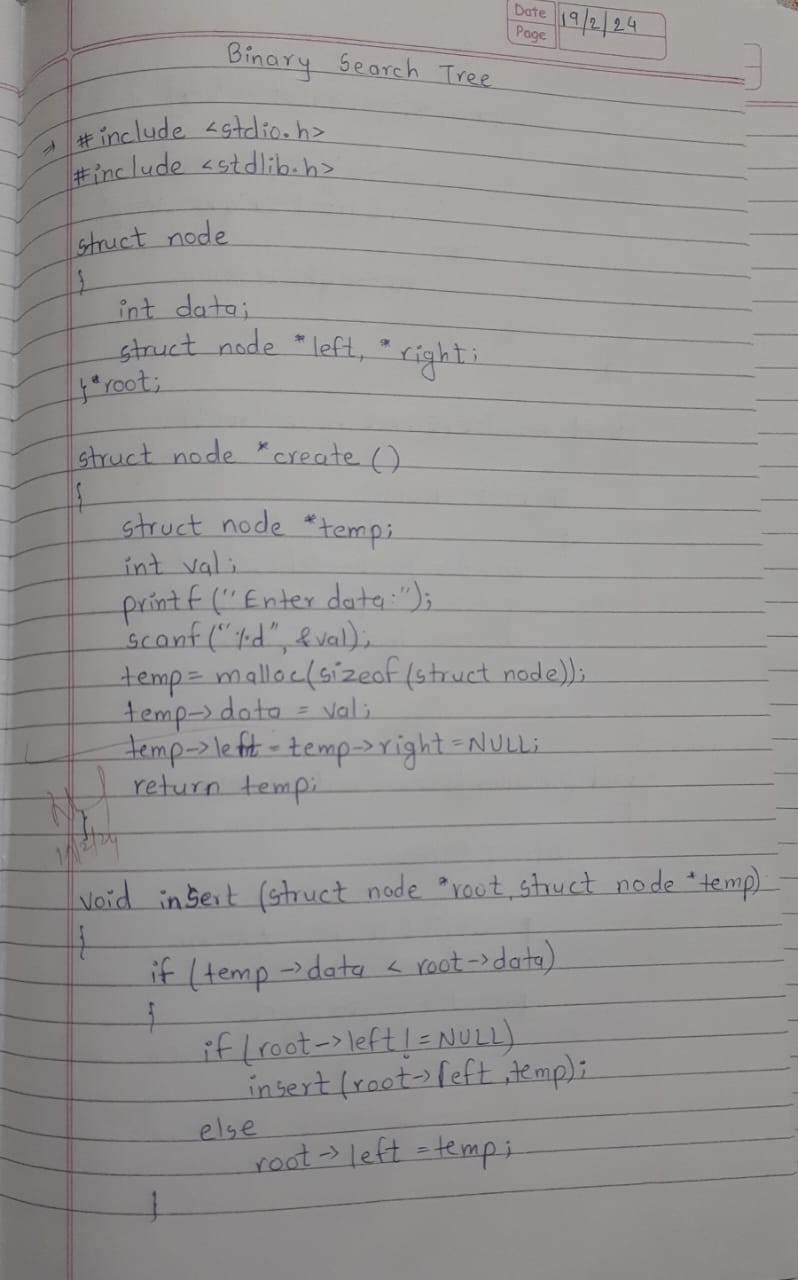
printf("INVALID INPUT");

}

}

}





**11a) 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++){

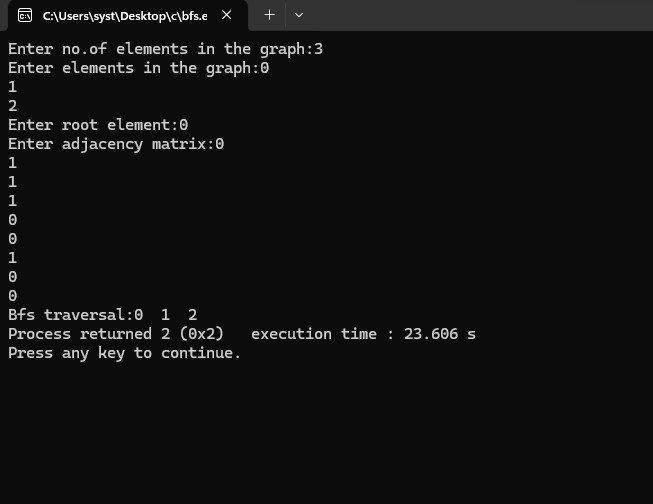
for(int j=0;j<n;j++){ scanf("%d",&a[i][j]);

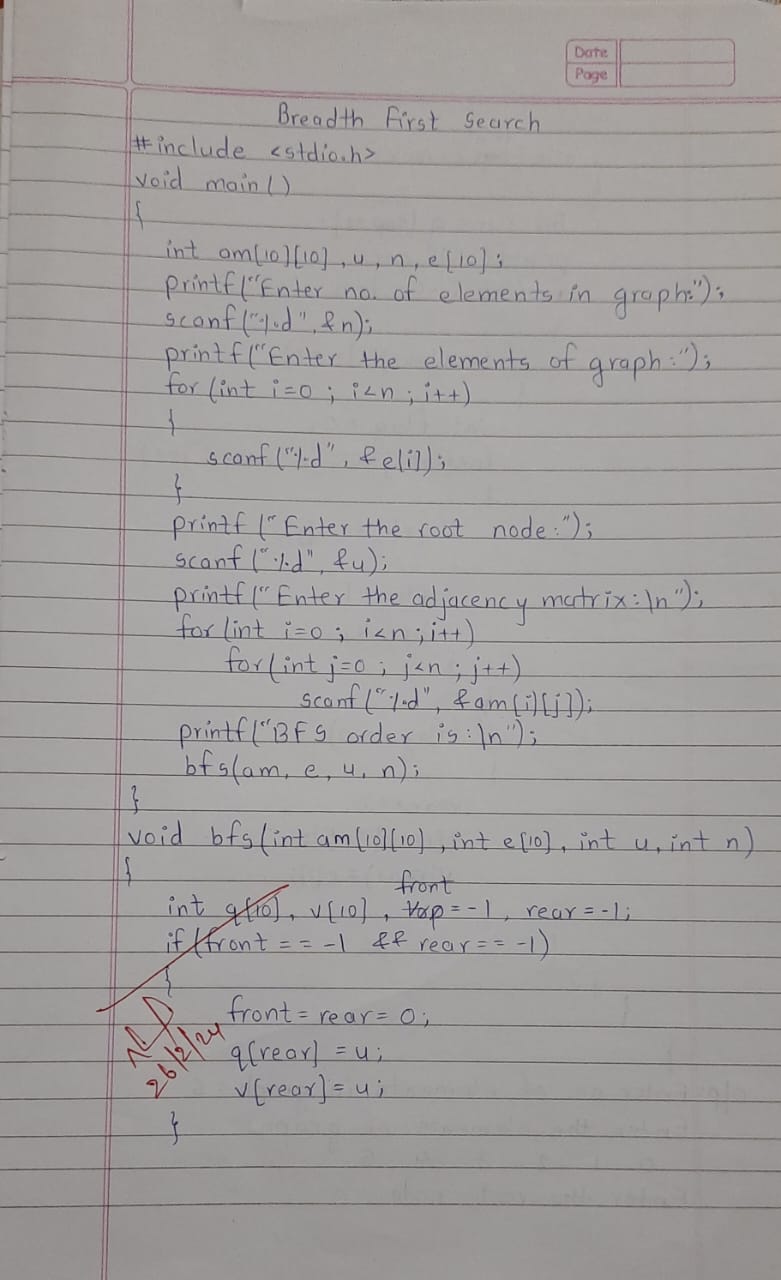
}

}

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

}





**11b) 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++;

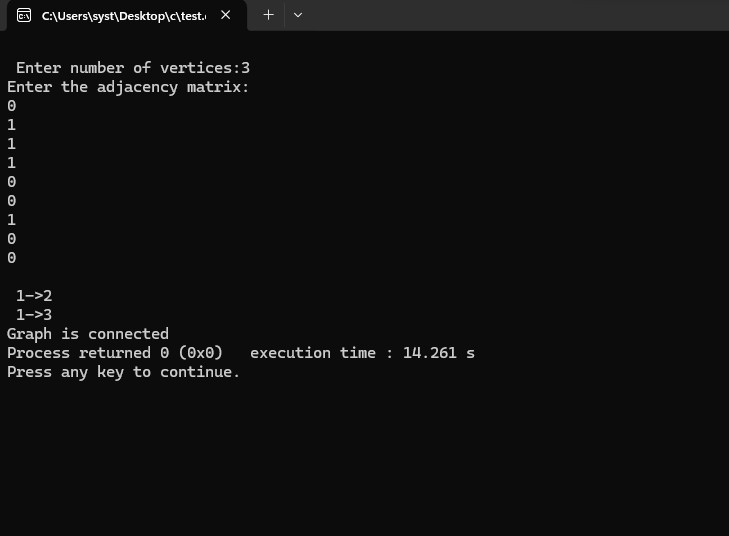
}

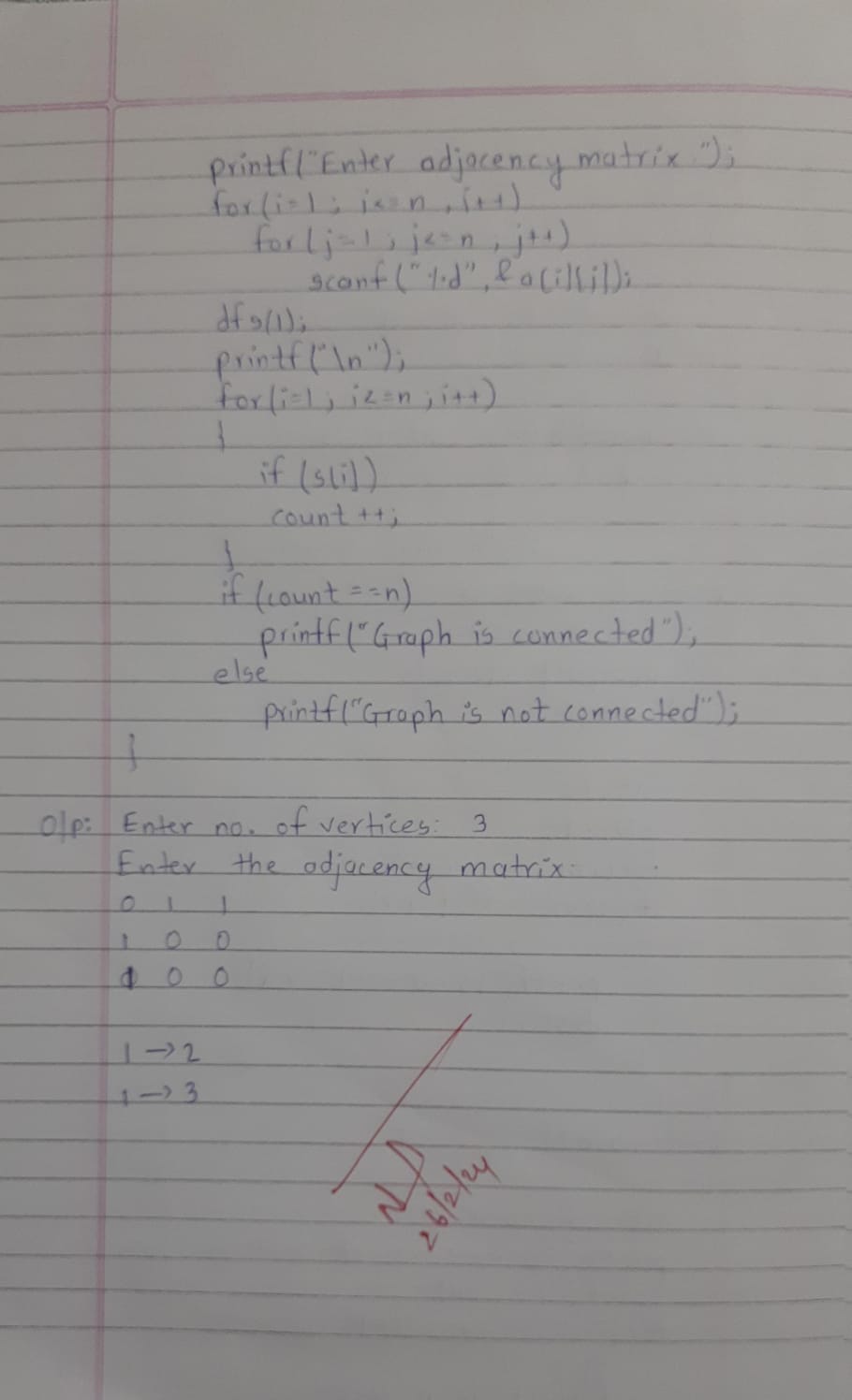
if(count==n)

printf("Graph is connected"); else

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

}





LEETCODE PROGRAM:DELETE NODE IN A 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;

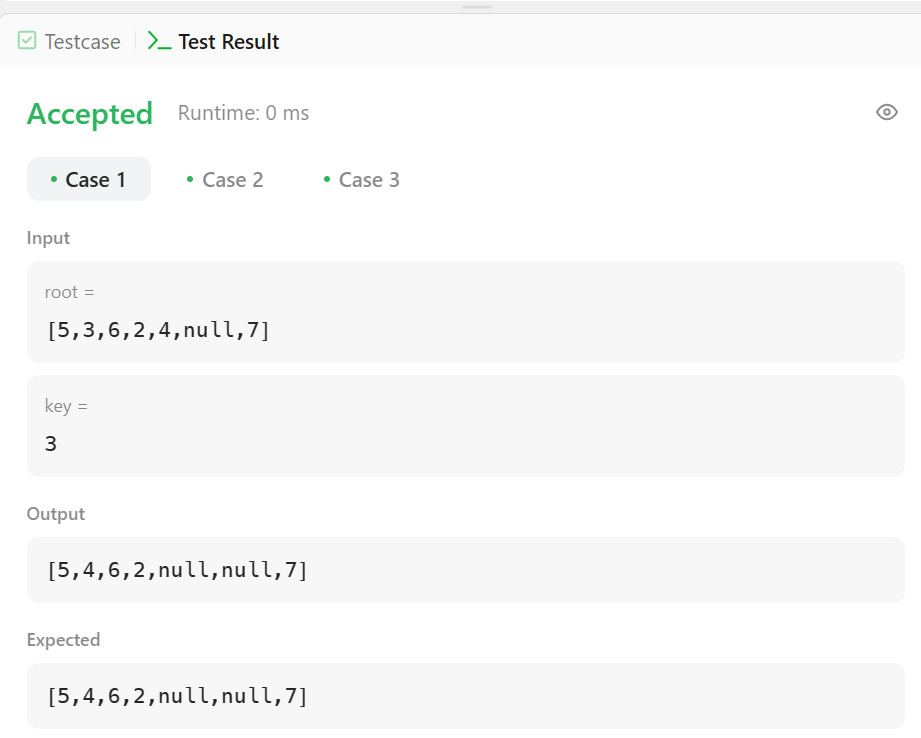
            root->right = deleteNode(root->right, temp->val);

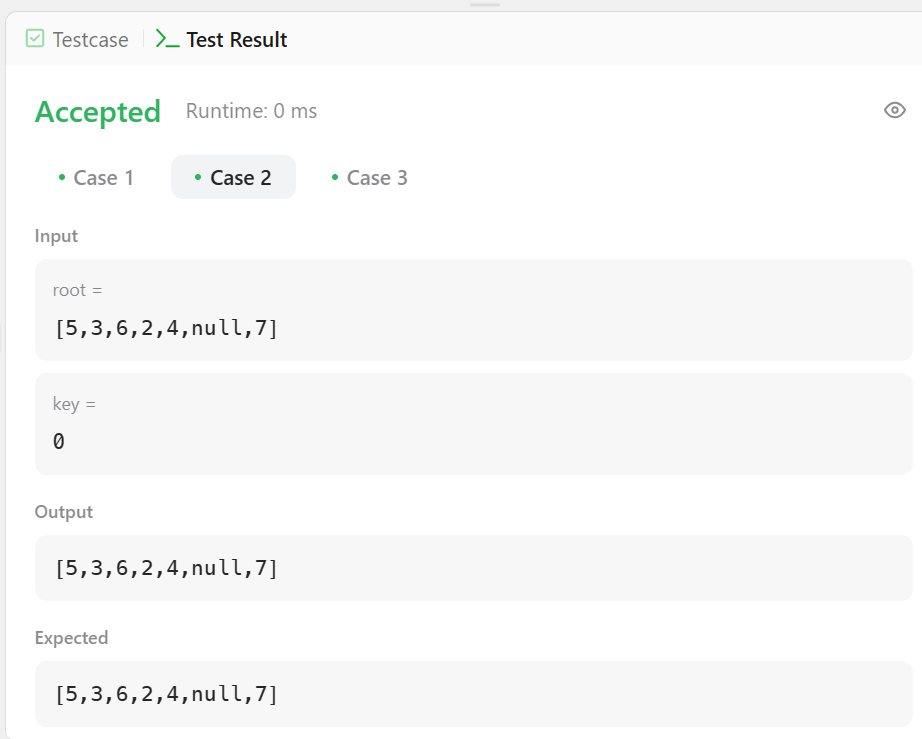
        }

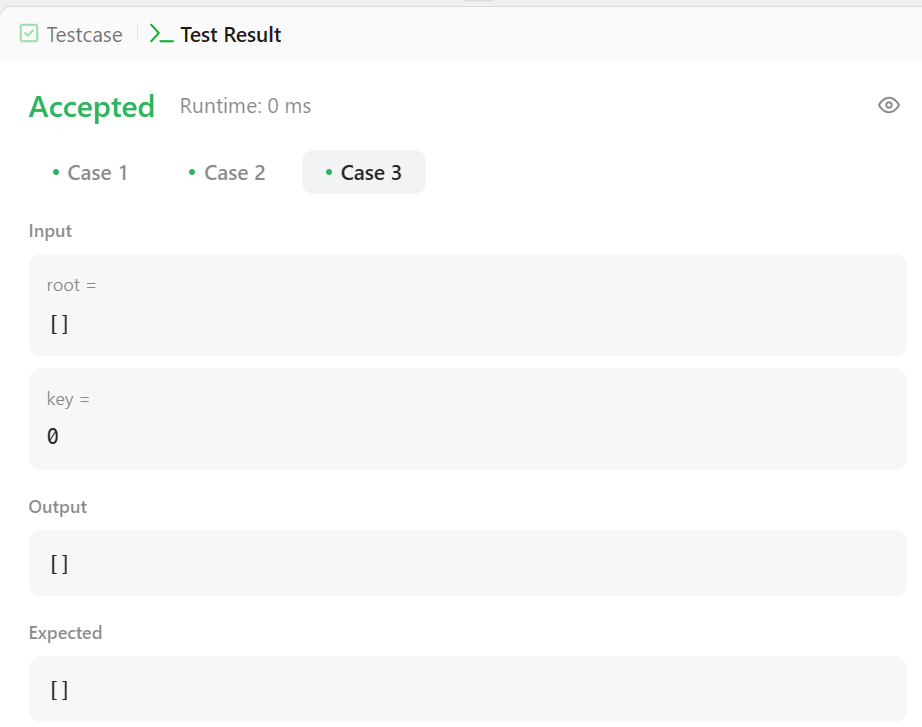
    }

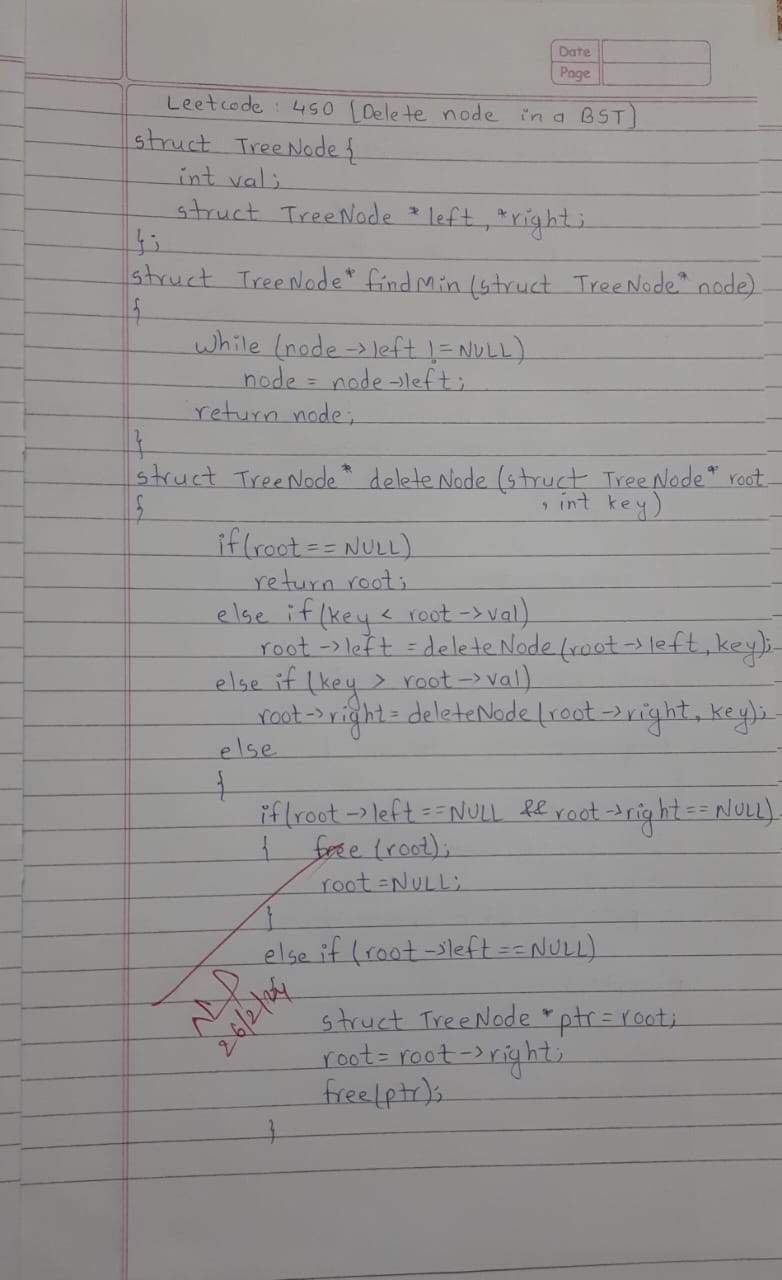
    return root;

}









LEETCODE PROGRAM: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;

            }

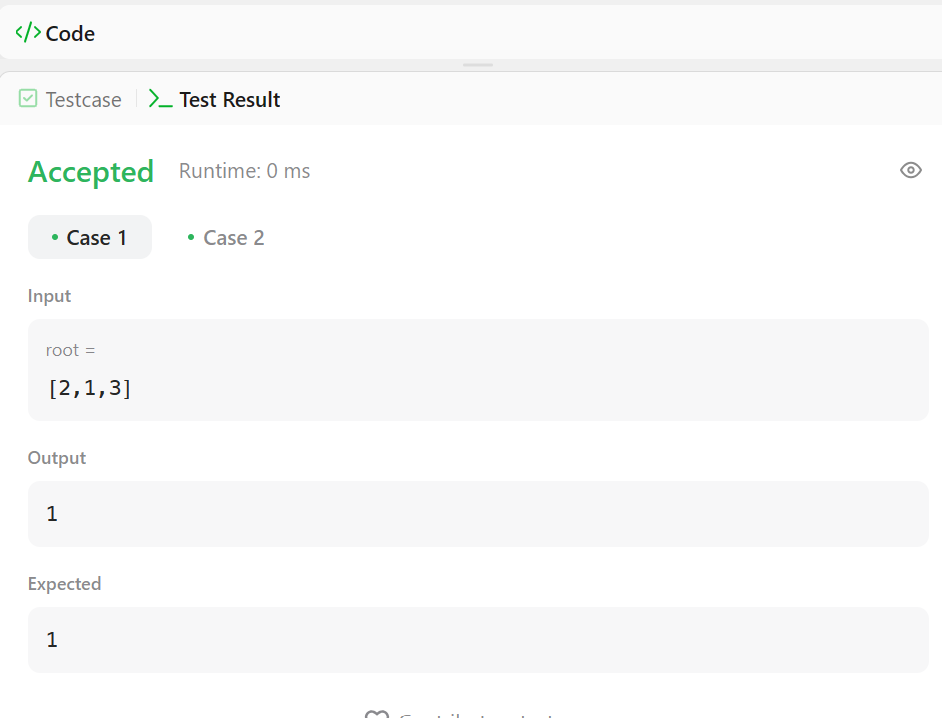
        }

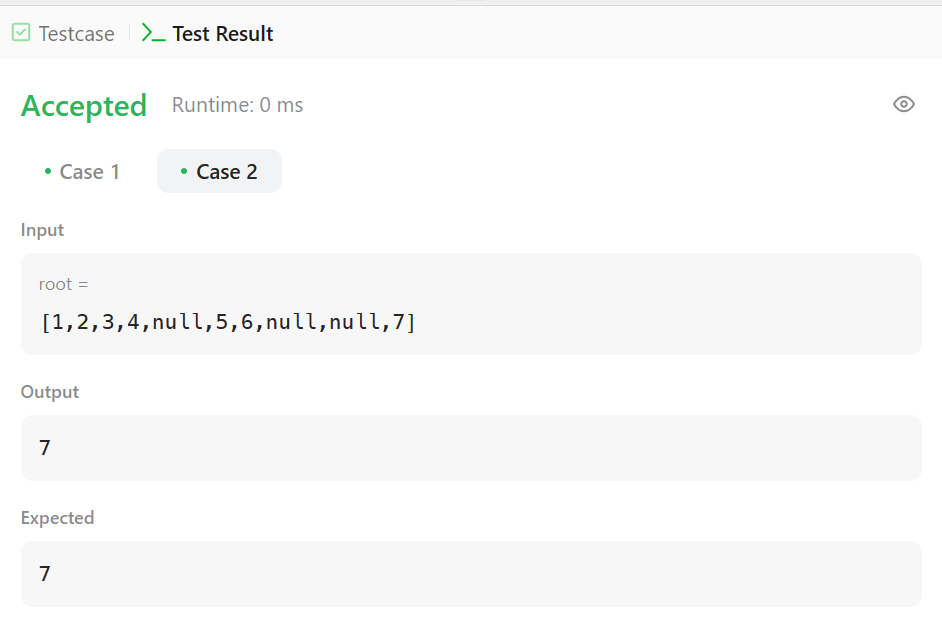
    }

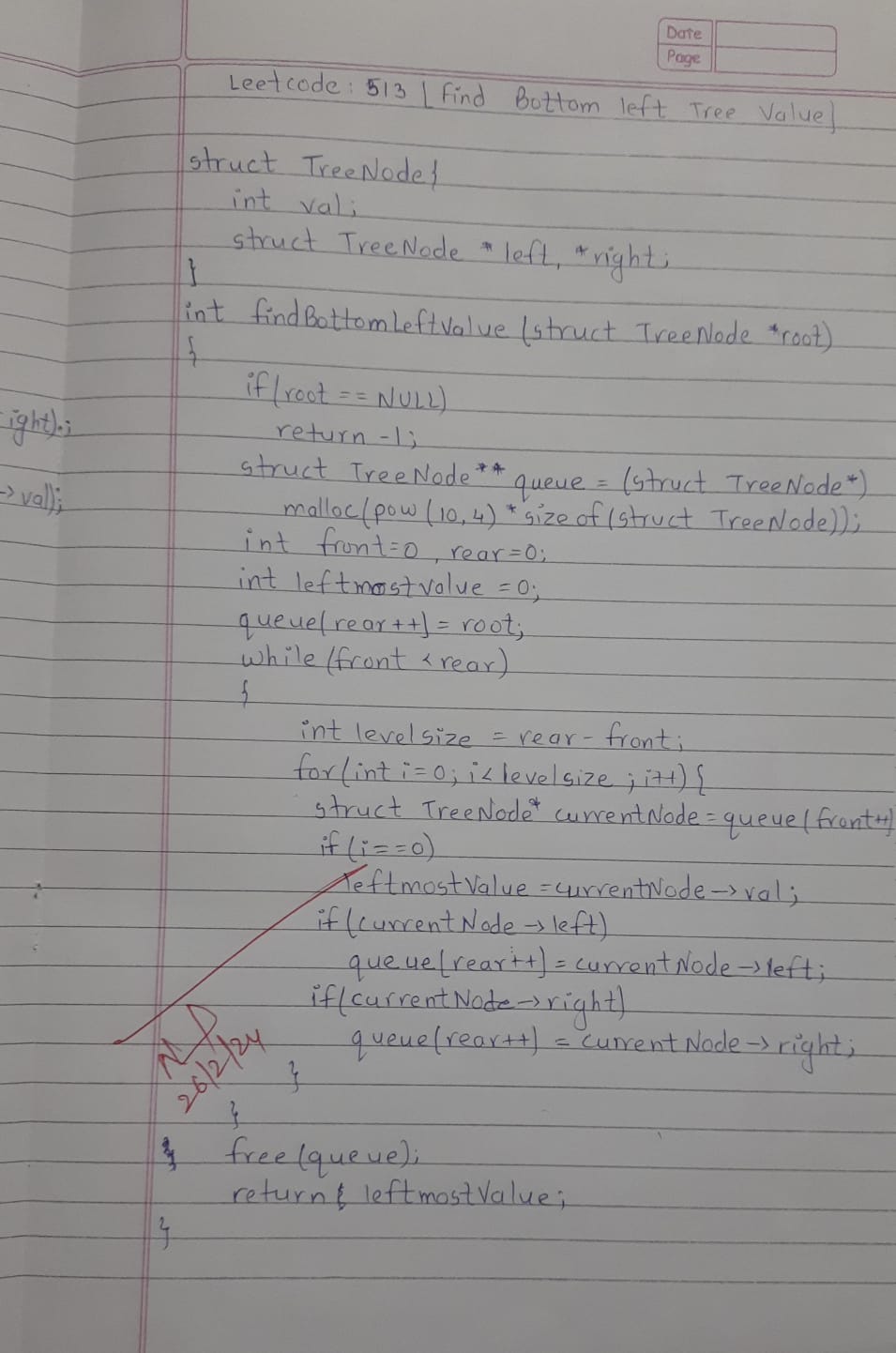
    free(queue);

    return leftmostValue;

}







**12)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;

}

