VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

DATA STRUCTURES IN C

Submitted by

SOHAN R (1BM23CS336)

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 Sep
2024-Jan 2025

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**DATA STRUCTURES IN C**" carried out by **SOHAN R(1BM23CS336),** 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 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of **Object- Oriented Java Programming Lab - (23CS3PCOOJ)** work prescribed for the said degree.

Geeta N

Associate Professor,
Department of CSE,
BMSCE, Bengaluru

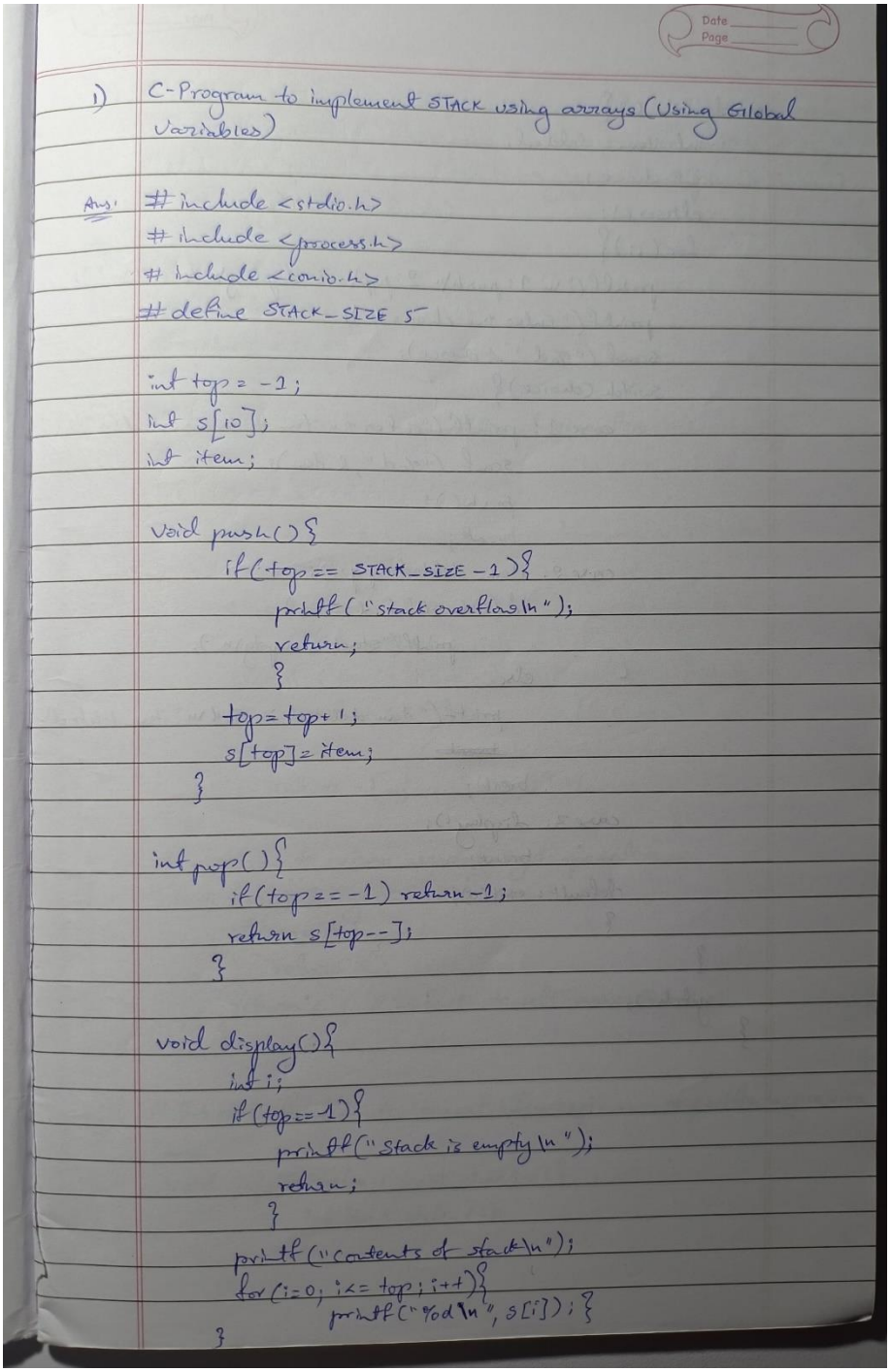
Dr. Kavitha Sooda

Professor and Head,
Department of CSE
BMSCE, Bengaluru

INDEX

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

Observation:

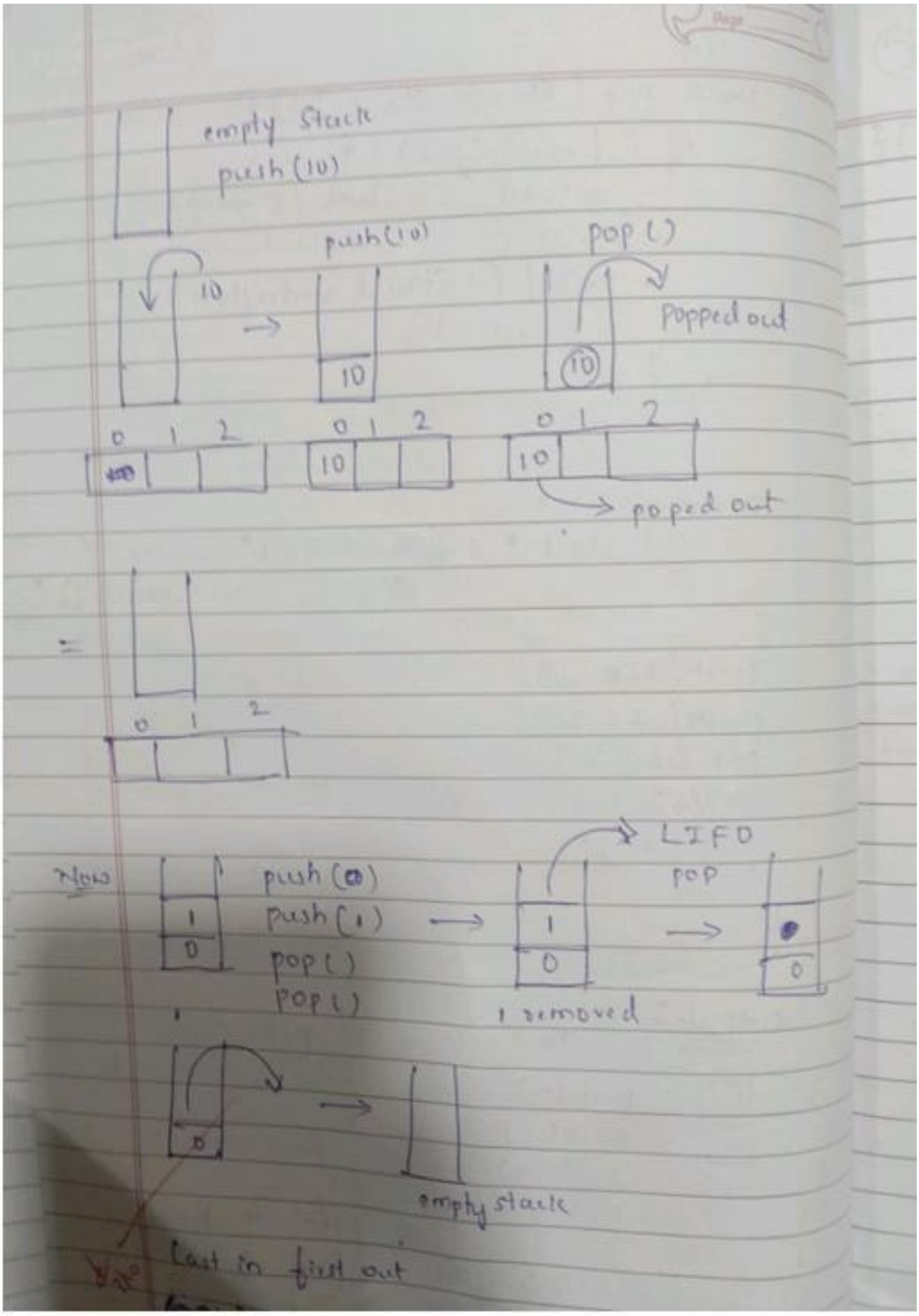


1) C-Program to implement STACK using arrays (Using Global Variables)

Ans:

```
#include <stdio.h>
#include <process.h>
#include <conio.h>
#define STACK_SIZE 5

int top = -1;
int s[10];
int item;

void push(){
    if(top == STACK_SIZE - 1){
        printf("stack overflow\n");
        return;
        }
    top = top+1;
    s[top] = item;
}

int pop(){
    if(top == -1) return -1;
    return s[top--];
}

void display(){
    int i;
    if(top==-1){
        printf("Stack is empty\n");
        return;
        }
    printf("contents of stack\n");
    for(i=0; i<=top; i++){
            printf("%d\n", s[i]); }
}
```

```
void main(){
    int item_deleted;
    int choice;
    clrscr();
    for(;;){
        printf("\n 1: push\n 2: pop\n 3: display\n 4: exit\n");
        printf("Enter the choice\n");
        scanf("%d", &choice);
        switch(choice){
            case 1: printf("enter the item to be inserted\n");
                    scanf("%d", &item);
                    push();
                    break;
            case 2: item_deleted = pop();
                    if(item_deleted == -1)
                        printf("stack is empty\n");
                    else
                        printf("item deleted is %d\n", item_deleted);
                        break
                    break;
            case 3: display();
                    break;
            default: exit(0);
            }
    }
    getch();
}
```


empty Stack
push (10)
push(10)
10
pop ()
Popped out
0 1 2
10
poped out
=
0 1 2
Now
push (0)
push (1)
pop ()
pop ()
LIFO
pop
1
0
1 removed
0
empty stack
Last in first out

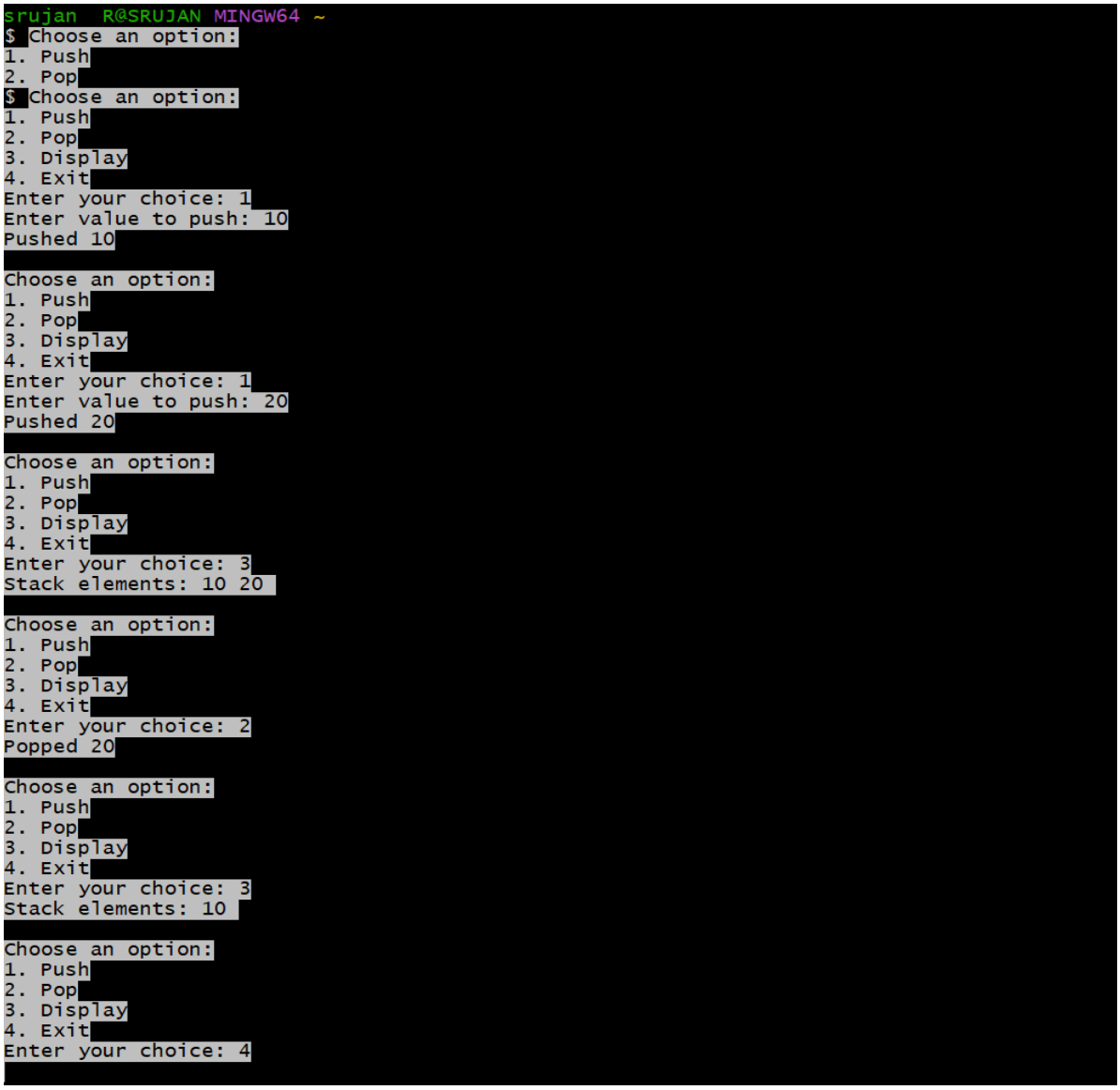
Code:

```
#include <stdio.h>
#define MAX 100

typedef struct {
    int arr[MAX];
    int top;
} Stack;

void push(Stack *s, int value) {
    if (s->top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    s->arr[++(s->top)] = value;
    printf("Pushed %d\n", value);
}

void pop(Stack *s) {
    if (s->top == -1) {
        printf("Stack Underflow\n");
        return;
    }
    printf("Popped %d\n", s->arr[(s->top)--]);
}

void display(Stack *s) {
    if (s->top == -1) {
        printf("Stack is Empty\n");
```

```
        return;
    }
    printf("Stack elements: ");
    for (int i = 0; i <= s->top; i++)
        printf("%d ", s->arr[i]);
    printf("\n");
}

int main() {
    Stack s;
    s.top = -1;
    int choice, value;

    while (1) {
        printf("\nChoose an option:\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&s, value);
                break;
            case 2:
                pop(&s);
                break;
            case 3:
```

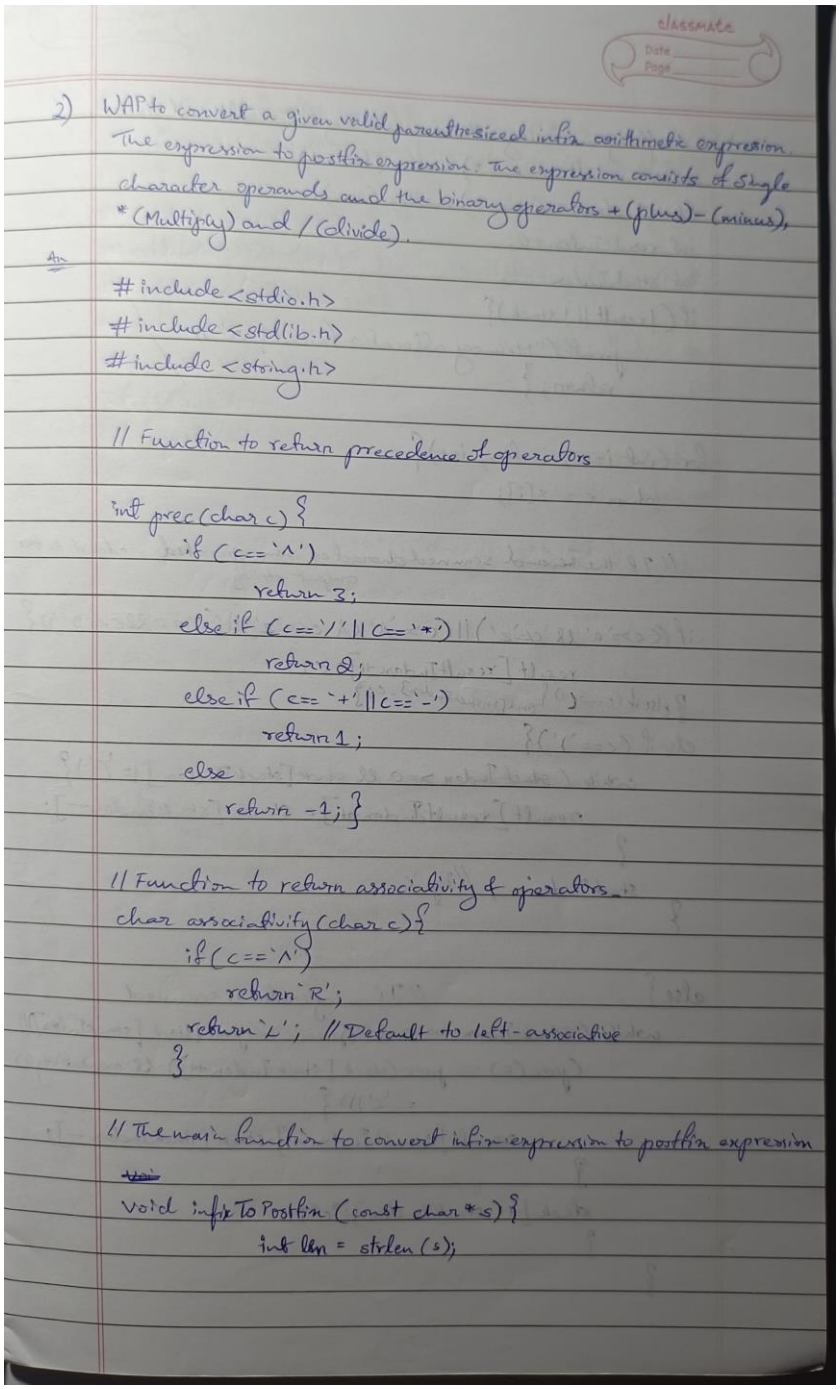
```
            display(&s);
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

Output:



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)

Observation:



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

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

// Function to return precedence of operators

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

// Function to return associativity of operators
char associativity(char c){
    if (c == '^')
        return 'R';
    return 'L'; // Default to left-associative
    }

// The main function to convert infix expression to postfix expression

void infixToPostfix (const char *s) {
        int len = strlen(s);
```

```
// Dynamically allocate memory for result and stack.
char* result = (char*) malloc (len + 1);
char* stack = (char*) malloc (len);
int resultIndex = 0;
int stackIndex = -1;
if (!result || !stack){
    printf ("Memory allocation failed! \n");
    return; }

for (int i=0; i<len; i++){
    char c = s[i];

// If the second scanned character is an operand, add it to the
                                    output string.

if ((c>='a' && c<='z') || (c>='A' && c<='Z') || (c>='0' && c<='9')){
        result [resultIndex++] = c; }
else if (c=='(') { stack[++stackIndex] = c; }
else if (c==')'){
    while (stackIndex >= 0 && stack[stackIndex] != '('){
        result [resultIndex++] = stack [stackIndex--];
    }
    stackIndex--; // Pop '('
}

else {                          // If an operator is scanned
    while (stackIndex >= 0 && (prec(c) < prec(stack [stackIndex]
        (prec(c) == prec(stack [stackIndex++]) && associativity
                                    == 'L'))) {
            result [resultIndex++] = stack [stackIndex--];
        }
        stack [++stackIndex] = c;
    }
}
```

```
while(stackIndex >= 0){    // Pop all the remaining elements from the stack
    result[resultIndex++] = stack[stackIndex--];
}

result[resultIndex] = '\0'    // Null terminate the result
    printf("%s\n", result);

// Free the allocated memory
free(result);
free(stack);}

int main()
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";

infixToPostfix(exp);
return 0;
}
```

Output:

abcd^e-fgh*+^*+i-

Seen

Gk

07/10

Code:

```
#include  <stdio.h>
#include <ctype.h>
#include <string.h>

#define MAX 100

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

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

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

int precedence(char c) {
   if (c == '+' || c == '-') return 1;
   if (c == '*' || c == '/') return 2;
   return 0;
}

void infixToPostfix(char* infix, char* postfix) {
   int i = 0, j = 0;
   char c, temp;

   while ((c = infix[i++]) != '\0') {
```

```
        if (isalnum(c)) {
            postfix[j++] = c;  // Append operand
        } else if (c == '(') {
            push(c);
        } else if (c == ')') {
            while (top != -1 && (temp = pop()) != '(') {
                postfix[j++] = temp;  // Pop till '('
            }
        } else {  // Operator
            while (top != -1 && precedence(stack[top]) >= precedence(c)) {
                postfix[j++] = pop();
            }
            push(c);
        }
    }

    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter a valid parenthesized infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
```

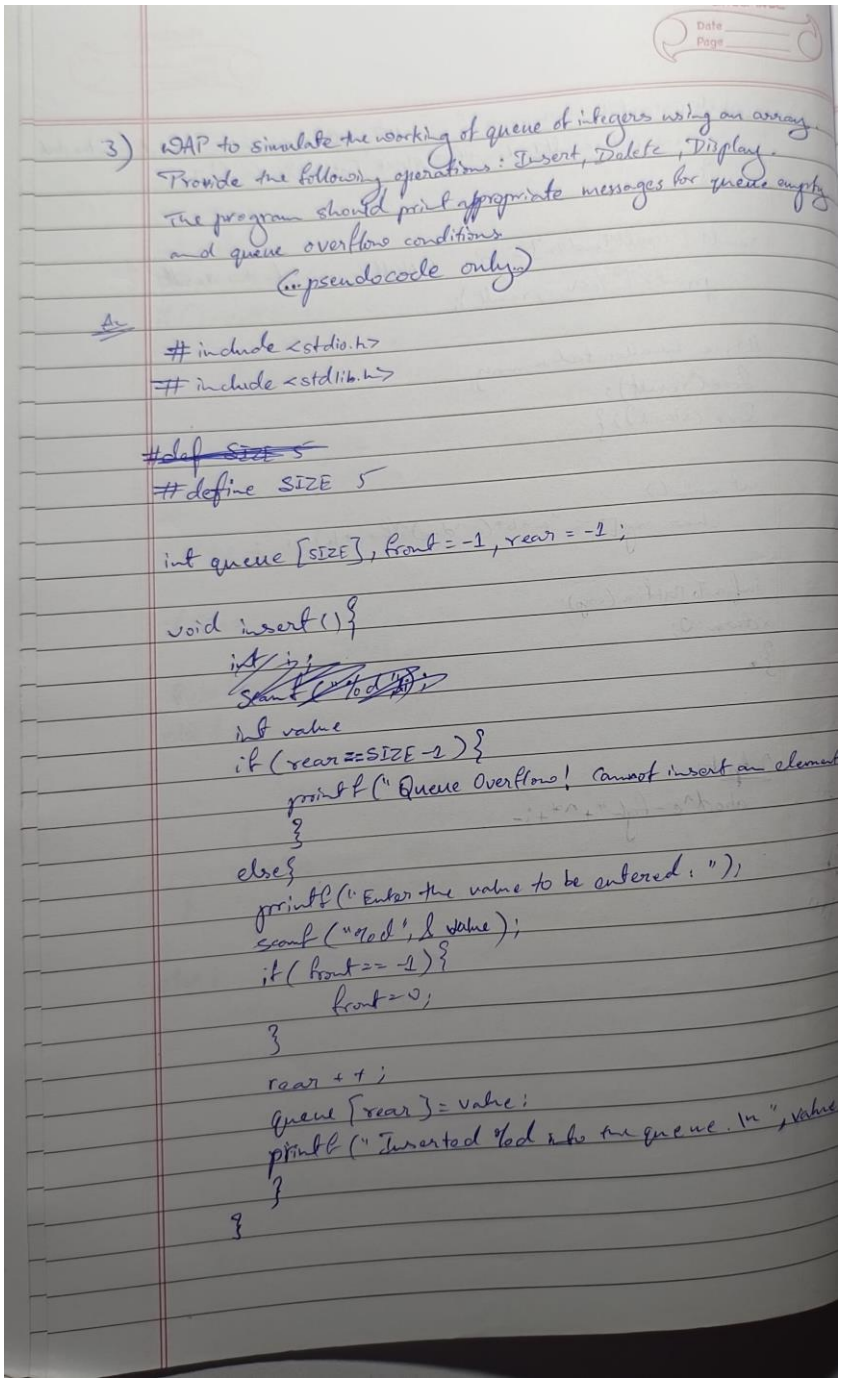
```
    printf("Postfix Expression: %s\n", postfix);

    return 0;
}
```

Output:

```
Enter a valid parenthesized infix expression: (A+B)*(c-D)
Postfix expression: AB+cD-*
```

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

Observation:



3) WAP to simulate the working of queue of integers using an array. Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions.
(...pseudocode only...)

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

#define SIZE 5

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

void insert(){
    int value
    if (rear==SIZE-1){
        printf("Queue Overflow! Cannot insert an element
    }
    else{
    printf("Enter the value to be entered: ");
    scanf("%d", &value);
    if(front==-1){
        front=0;
    }
    rear++;
    queue[rear]=value;
    printf("Inserted %d to the queue.\n", value
    }
}
```

```
void delete(){
    if(front==-1 || front > rear){
        printf("Queue Underflow | Queue is empty\n")
    }
    else{
        printf("Deleted %d from the queue\n", queue[front]
        front++;
        if(front > rear){
            front = rear = -1;
        }
    }
}

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

Void main(){
    .
    .
    .
    (Big switch case
}
```

Output:

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1
Enter the value to insert: 2
Inserted 2 into the queue

Enter your choice: 1
Enter the value to insert: 4
Inserted 4 into the queue.

Enter your choice: 1
Enter the value to insert: 6
Inserted 6 into the queue.

Enter your choice: 1
Enter the value to insert: 8
Inserted 8 into the queue.

Enter your choice: 1
Enter the value to insert: 10
Inserted 10 into the queue.

Enter your choice: 1
Queue overflow! Cannot insert element

*

Enter your choice : 3

Queue elements : 2 4 6 8 10

*

Enter your choice : 2

Deleted 2 from the queue

*

Enter your choice : 2

Deleted 4 from the queue

*

Enter your choice : 2

Deleted 6 from the queue

*

Enter your choice : 2

Deleted 8 from the queue

*

Enter your choice : 2

Deleted 10 from the queue

*

Enter your choice : 2

Queue Underflow! Queue is empty.

*

Enter your choice : 4

Exiting...

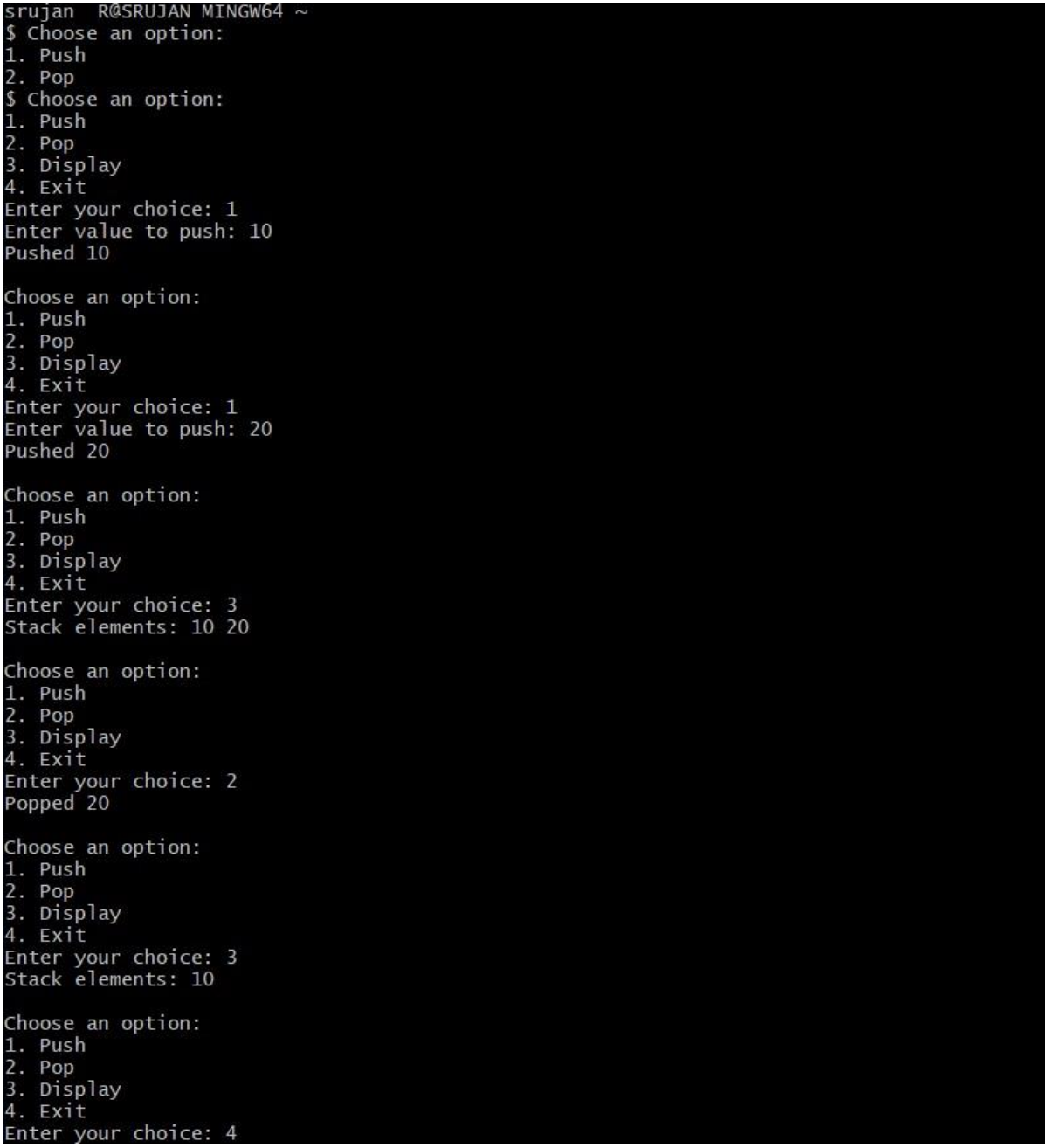
Code:

```
#include <stdio.h>
#define MAX 5

// Linear Queue
int queue[MAX], front = -1, rear = -1;

void insert(int val) {
   if (rear == MAX - 1) {
      printf("Queue Overflow\n");
   } else {
      if (front == -1) front = 0;
      queue[++rear] = val;
      printf("Inserted %d\n", val);
   }
}

void delete() {
   if (front == -1 || front > rear) {
      printf("Queue Underflow\n");
   } else {
      printf("Deleted %d\n", queue[front++]);
      if (front > rear) front = rear = -1;
   }
}

void display() {
   if (front == -1) {
      printf("Queue is Empty\n");
```

```
    } else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("\nLinear Queue Operations:\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
```

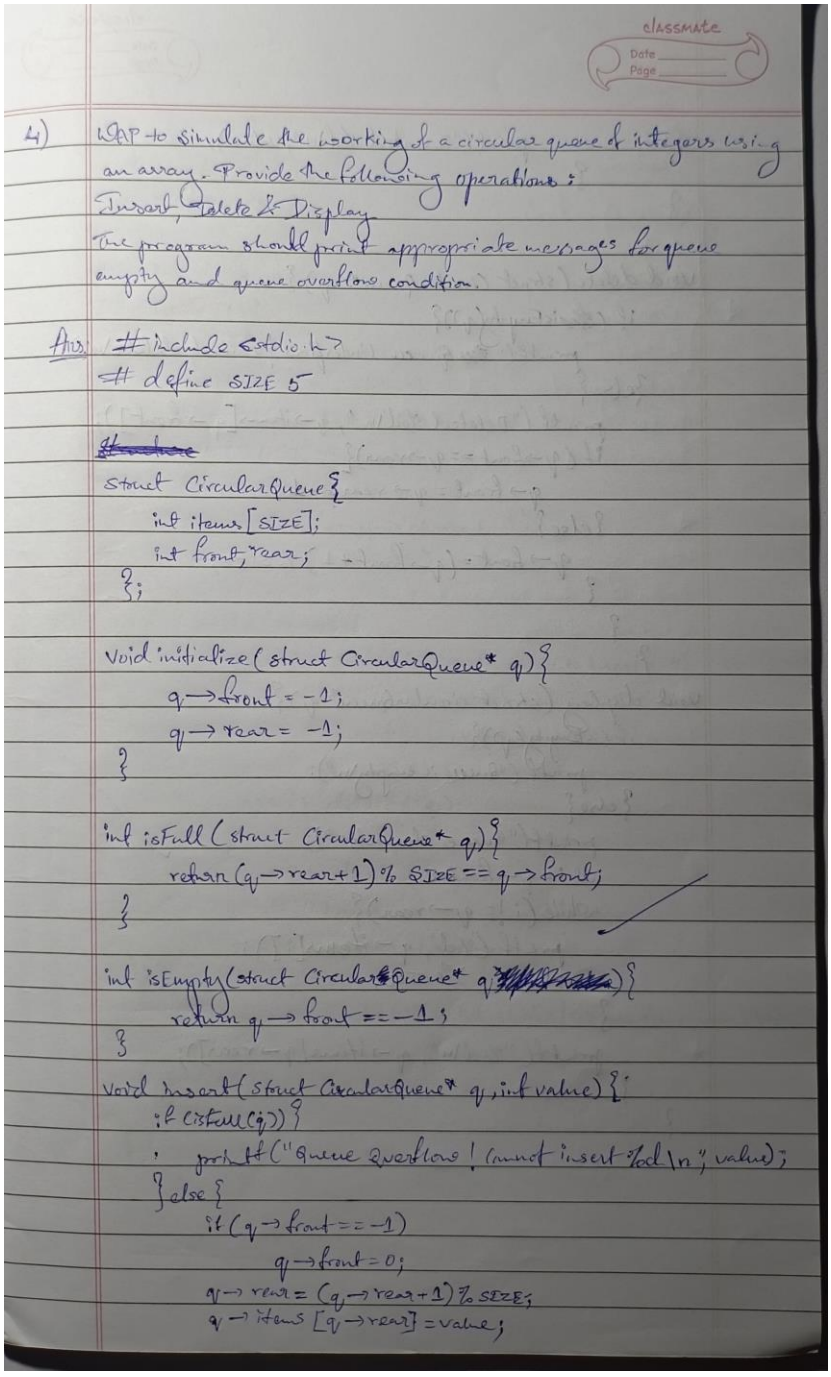
```
    }
}
```

Output:


srujan R@SRUJAN MINGW64 ~
$ Choose an option:
1. Push
2. Pop
$ Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10
Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 20
Pushed 20
Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 10 20
Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 20
Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 10
Choose an option:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4

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

Observation:



4) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:
Insert, Delete & Display
The program should print appropriate messages for queue empty and queue overflow condition.

Ans:

```
#include <stdio.h>
#define SIZE 5

struct CircularQueue{
    int items[SIZE];
    int front, rear;
};

void initialize(struct CircularQueue* q){
    q->front = -1;
    q->rear = -1;
}

int isFull(struct CircularQueue* q){
    return (q->rear+1) % SIZE == q->front;
}

int isEmpty(struct CircularQueue* q){
    return q->front == -1;
}

void insert(struct CircularQueue* q, int value){
    if(isFull(q)){
        printf("Queue overflow! Cannot insert %d\n", value);
    }else{
        if(q->front == -1)
            q->front = 0;
        q->rear = (q->rear+1) % SIZE;
        q->items[q->rear] = value;
```

```
        printf("Inserted %d\n", value);
    }
}

void delete(struct CircularQueue* q){
    if (isEmpty(q)){
        printf("Queue Underflow! cannot delete\n");
    }else{
        printf("Deleted %d\n", q->items[q->front]);
        if(q->front == q->rear){
            q->front = q->rear = -1;
        }else{
            q->front = (q->front+1) % SIZE;
        }
    }
}
void display(struct circularQueue* q){
    if(isEmpty(q)){
        printf("Queue is empty\n");
    }else{
        printf("Queue elements: ");
        int i = q->front;
        while(i != q->rear){
            printf("%d", q->items[i]);
            i = (i+1) % SIZE;
        }
        printf("%d\n", q->items[q->rear]);
    }
}
```

```
int main (){
    struct CircularQueue q;
    initialize(&q);

    int choice, value;
    while (1){
        printf("\n Queue Operations: \n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1: printf("Enter the value to insert: ");
                    scanf("%d", &value);
                    insert(&q, value);
                    break;
            case 2: delete(&q);
                    break;
            case 3: display(&q);
                    break;
            case 4: printf("Exiting....\n");
                    return 0;
            default: printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output:-

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1
Enter the value to insert: 1
inserted 1

Enter your choice: 1
Enter the value to insert: 2
inserted 2

Enter your choice: 1
Enter the value to insert: 3
inserted 3

Enter your choice: 1
Enter the value to insert: 4
inserted 4

Enter your choice: 1
Enter the value to insert: 6
Queue Overflow! cannot insert 6

Enter your choice: 3
Queue elements: 1 2 3 4 5

Enter your choice: 2
deleted 1.

Enter your choice: 2
deleted 2.

Enter your choice: 1
Enter the value to insert: 6
inserted 6

Enter your choice: 1
Enter the value to insert: 7
inserted 7.

Enter your choice: 3
Queue elements: 3 4 5 6 7

Enter your choice: 2
deleted 3

Enter your choice: 2
deleted 4

Date
Page

*

Enter your choice: 2
deleted 5

*

Enter your choice: 2
deleted 6

*

Enter your choice: 2
deleted 7

*

Enter your choice: 2
Queue Underflow! cannot delete

*

Enter your choice: 4
Exiting...

== Code execution successful. ==

Code:

```
#include <stdio.h>
#define MAX 5

// Circular Queue
int queue[MAX], front = -1, rear = -1;

void insert(int val) {
   if ((front == 0 && rear == MAX - 1) || (front == rear + 1)) {
      printf("Queue Overflow\n");
   } else {
      if (front == -1) {
         front = rear = 0;
      } else if (rear == MAX - 1) {
         rear = 0;
      } else {
         rear++;
      }
      queue[rear] = val;
      printf("Inserted %d\n", val);
   }
}

void delete() {
   if (front == -1) {
      printf("Queue Underflow\n");
   } else {
```

```
        printf("Deleted %d\n", queue[front]);
        if (front == rear) {
            front = rear = -1;
        } else if (front == MAX - 1) {
            front = 0;
        } else {
            front++;
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is Empty\n");
    } else {
        printf("Queue elements: ");
        if (rear >= front) {
            for (int i = front; i <= rear; i++) printf("%d ", queue[i]);
        } else {
            for (int i = front; i < MAX; i++) printf("%d ", queue[i]);
            for (int i = 0; i <= rear; i++) printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;
    while (1) {
```

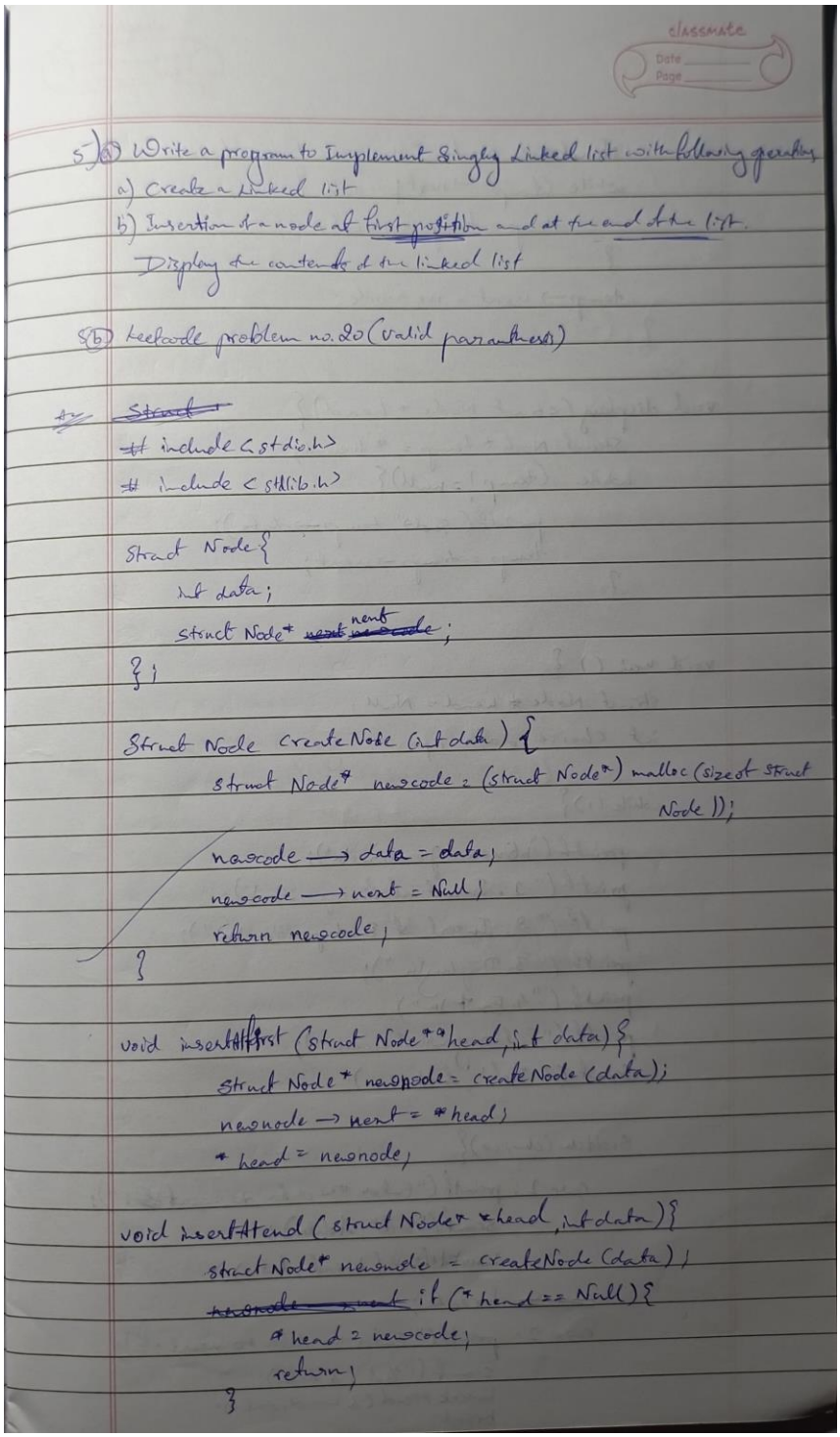
```
        printf("\nCircular Queue Operations:\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 10
Inserted 10

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 20
Inserted 20

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 30
Inserted 30

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20 30

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 10

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20 30

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 40
Inserted 40
```

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

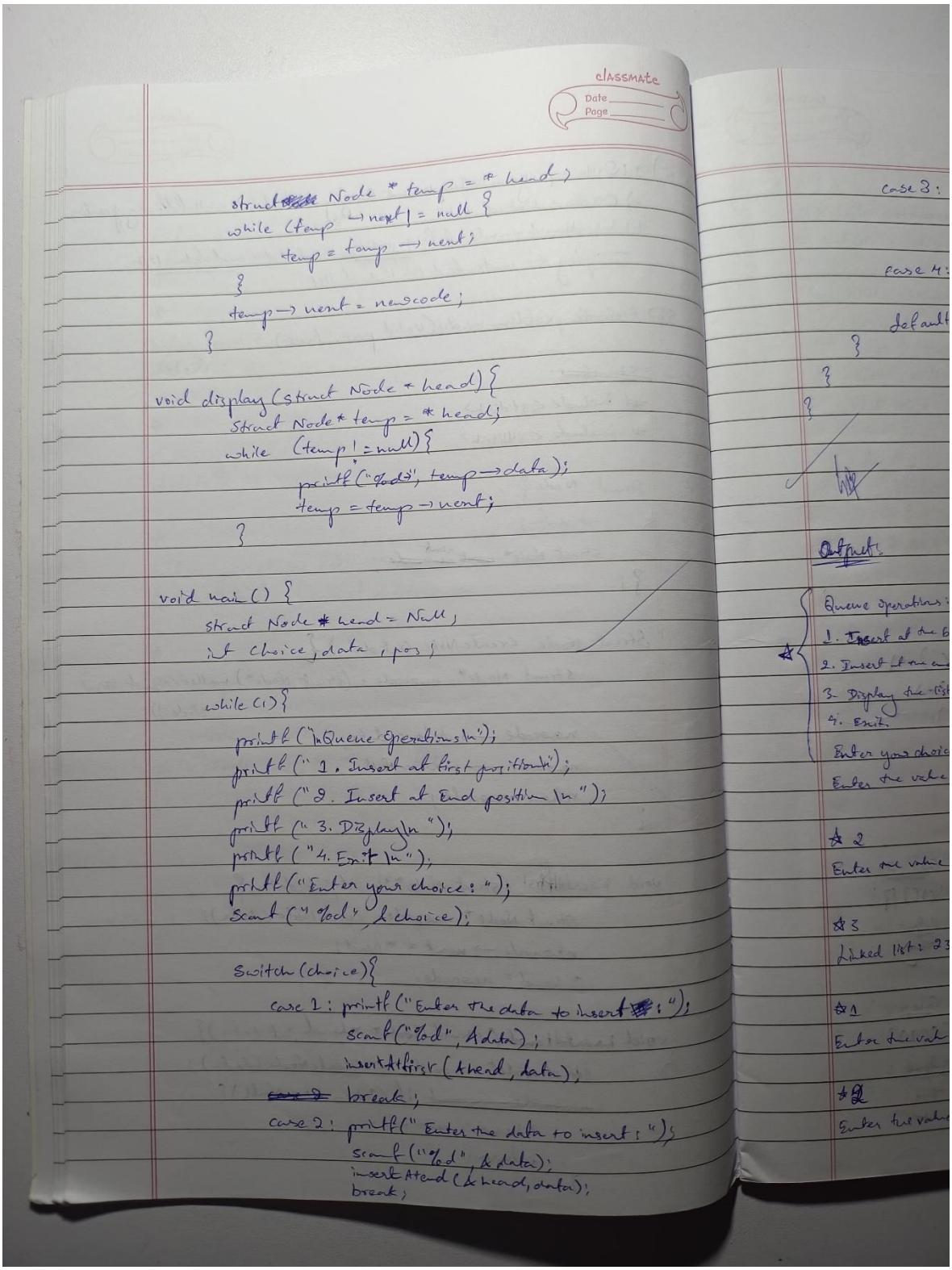
Observation:



5) (a) Write a program to Implement Singly Linked list with following operations
a) Create a linked list
b) Insertion of a node at first position and at the end of the list.
Display the contents of the linked list

(b) leetcode problem no. 20 (valid parantheses)

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

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

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

void insertAtfirst(struct Node** head, int data){
    struct Node* newnode = createNode(data);
    newnode -> next = *head;
    *head = newnode;

void insertAtend(struct Node* *head, int data){
    struct Node* newnode = createNode(data);
    if(*head == Null){
        *head = newcode;
        return;
    }
```



```
struct Node * temp = * head;
while (temp -> next != null {
    temp = temp -> next;
}
temp -> next = newcode;
}

void display (struct Node * head) {
    Struct Node * temp = * head;
    while (temp != null) {
        printf ("%d", temp -> data);
        temp = temp -> next;
    }

void main () {
    struct Node * head = Null;
    int choice, data, pos;

    while (1) {
        printf ("\nQueue operations\n");
        printf (" 1. Insert at first position\n");
        printf ("2. Insert at End position \n");
        printf (" 3. Display\n");
        printf ("4. Exit \n");
        printf ("Enter your choice: ");
        scanf ("%d", & choice);

        switch (choice) {
            case 1: printf ("Enter the data to insert: ");
                    scanf ("%d", & data);
                    insertAtfirst (& head, data);
                    break;
            case 2: printf ("Enter the data to insert: ");
                    scanf ("%d", & data);
                    insertAtend (& head, data);
                    break;
```

```
case 3:

case 4:

default

}
}
}

Output:

Queue operations
1. Insert at the
2. Insert at the
3. Display the list
4. Exit
Enter your choice
Enter the value

2
Enter the value

3
Linked list: 2

1
Enter the val

2
Enter the valu
```

```
        case 3: printf("Queue elements: ");
                display(head);
                break;
        case 4: exit(0);
                break;
        default: printf("Invalid choice.");
    }
  }
}
```

Output:

Queue operations:
1. Insert at the beginning
2. Insert at the end
3. Display the list
4. Exit.

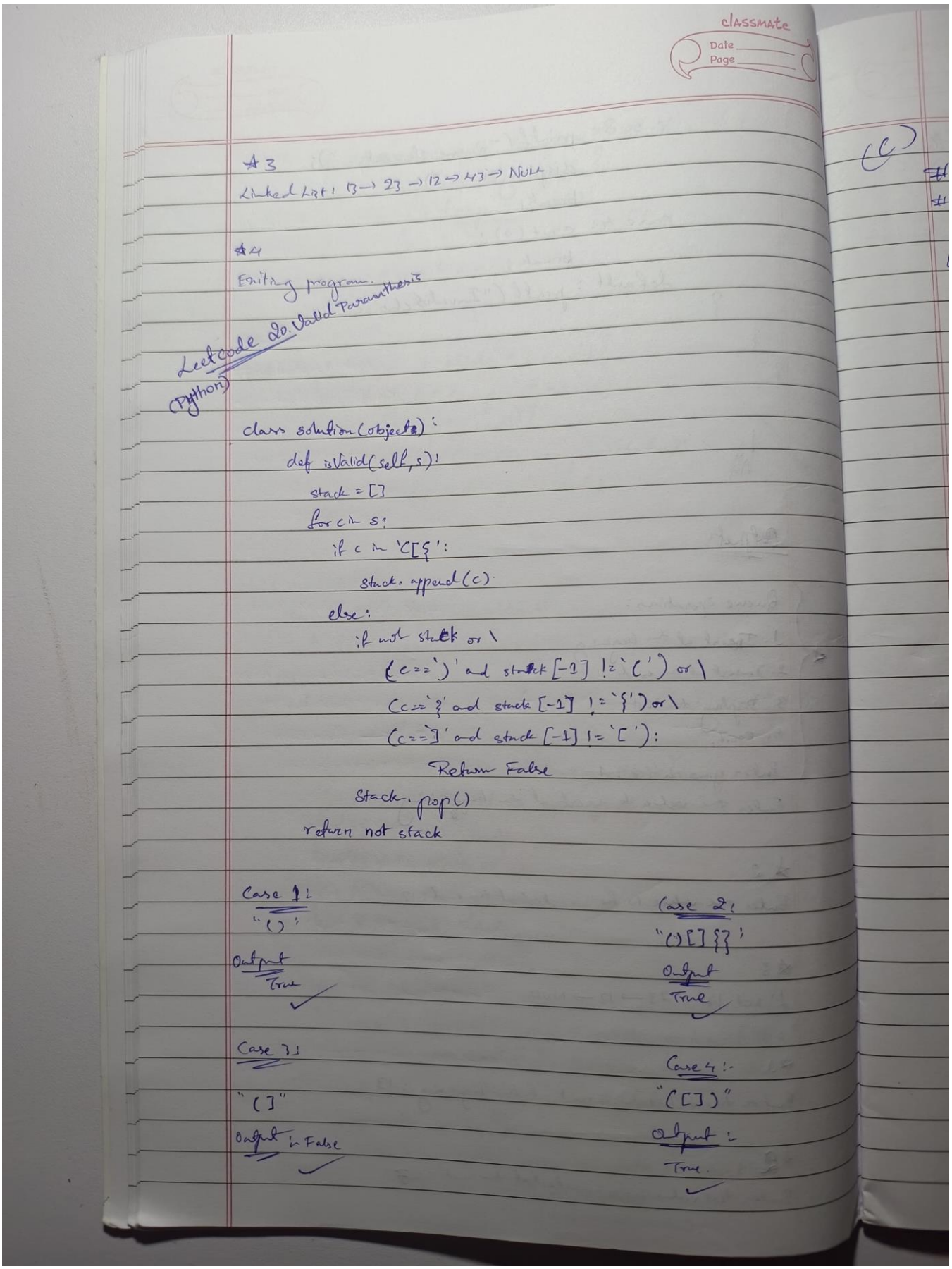
Enter your choice: 1
Enter the value to insert at the beginning: 23

☆ 2
Enter the value to be inserted at the end: 12

☆ 3
Linked list: 23 → 12 → NULL

☆ 1
Enter the value to insert at the beginning: 13

☆ 2
Enter the value to be inserted at the end: 45


classmate
Date
Page
A3
Linked List: 13 → 23 → 12 → 43 → NULL
A4
Exiting program.
Leetcode 20. Valid Parenthesis
(Python)
class solution(object):
def isValid(self,s):
stack = []
for c in s:
if c in '([{':
stack.append(c)
else:
if not stack or \
(c==')' and stack[-1] !='(') or \
(c=='}' and stack[-1] !='{') or \
(c==']' and stack[-1] !='['):
Return False
Stack.pop()
return not stack
Case 1:
"()"
Output
True
Case 2:
"()[]{}"
Output
True
Case 3:
"(]"
Output : False
Case 4:
"([])"
Output :
True.

(c)

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

bool isValid(char* s){
    int length = 0;
    while (s[length] != '\0') length++;
    char* stack = (char*)malloc(length * sizeof(char));
    int top = -1;

    for (int i=0; i<length; i++) {
        if (s[i] == '(' || s[i] == '{' || s[i] == '[') {
            stack[++top] = s[i];
        } else {
            if (top == -1 ||
                (s[i] == ')' && stack[top] != '(') ||
                (s[i] == '}' && stack[top] != '{') ||
                (s[i] == ']' && stack[top] != '[')) {
                free(stack);
                return false;
            }
            top--;
        }
    }
    bool isValid = (top == -1);
    free(stack);
    return isValid;
}
```

Code:

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

// Node structure for Singly Linked List
struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;

// Create a linked list
void createLinkedList(int n) {
    struct Node *newNode, *temp;
    int data, i;

    for (i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);

        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = NULL;

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

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

// Insert at the beginning
void insertAtBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("Node inserted at the beginning.\n");
}

// Insert at a specific position
void insertAtPosition(int data, int position) {
    struct Node *newNode, *temp;
    int i;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;

    if (position == 1) {
        newNode->next = head;
        head = newNode;
    } else {
```

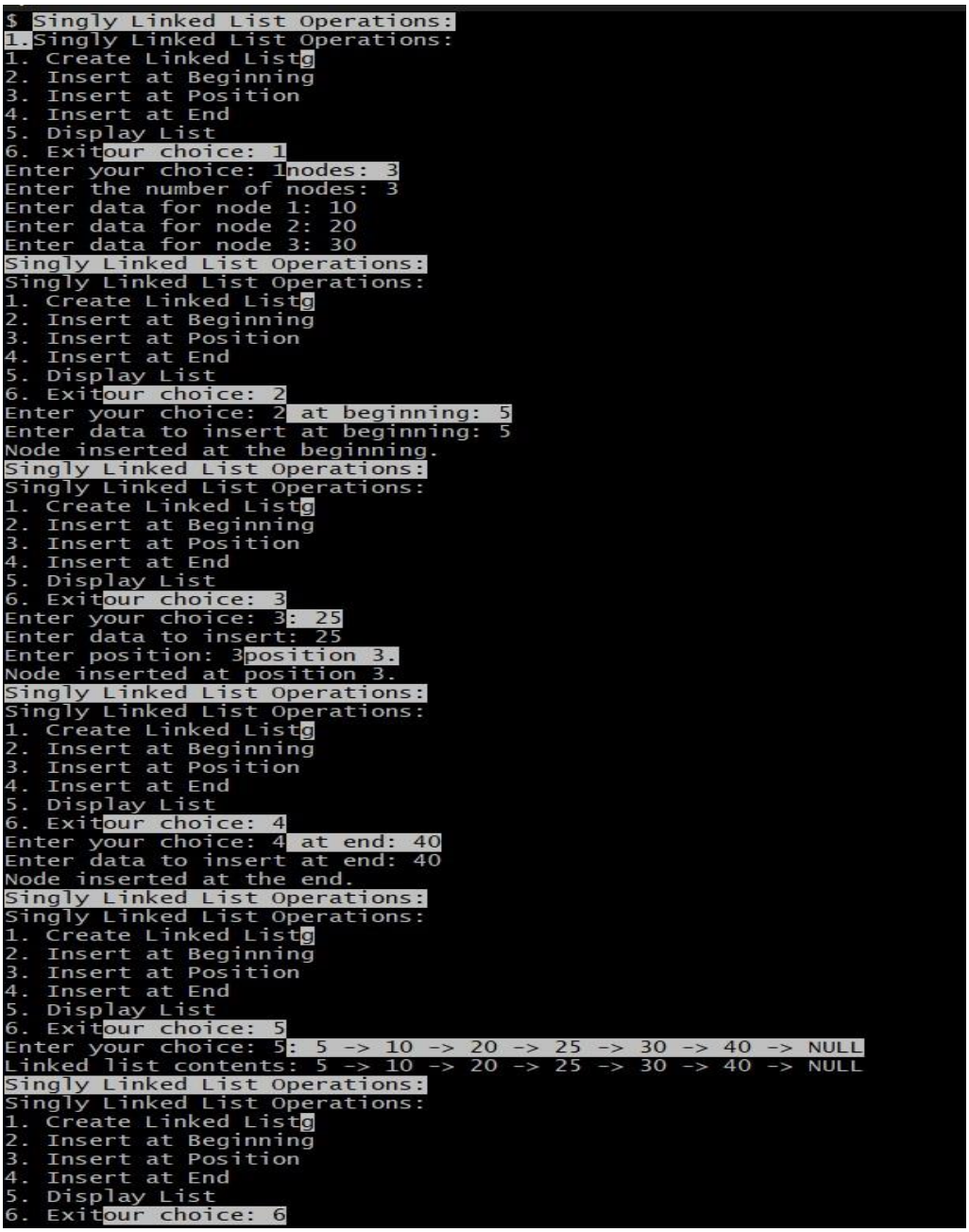
```
        temp = head;
        for (i = 1; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }

        if (temp != NULL) {
            newNode->next = temp->next;
            temp->next = newNode;
            printf("Node inserted at position %d.\n", position);
        } else {
            printf("Position out of range.\n");
        }
    }
}

// Insert at the end
void insertAtEnd(int data) {
    struct Node *newNode, *temp;

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

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

```
        }
        temp->next = newNode;
    }
    printf("Node inserted at the end.\n");
}

// Display the linked list
void displayList() {
    struct Node* temp = head;

    if (head == NULL) {
        printf("The list is empty.\n");
    } else {
        printf("Linked list contents: ");
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int choice, data, position, n;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Create Linked List\n2. Insert at Beginning\n3. Insert at Position\n4. Insert at
End\n5. Display List\n6. Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the number of nodes: ");
        scanf("%d", &n);
        createLinkedList(n);
        break;
    case 2:
        printf("Enter data to insert at beginning: ");
        scanf("%d", &data);
        insertAtBeginning(data);
        break;
    case 3:
        printf("Enter data to insert: ");
        scanf("%d", &data);
        printf("Enter position: ");
        scanf("%d", &position);
        insertAtPosition(data, position);
        break;
    case 4:
        printf("Enter data to insert at end: ");
        scanf("%d", &data);
        insertAtEnd(data);
        break;
    case 5:
        displayList();
        break;
    case 6:
```

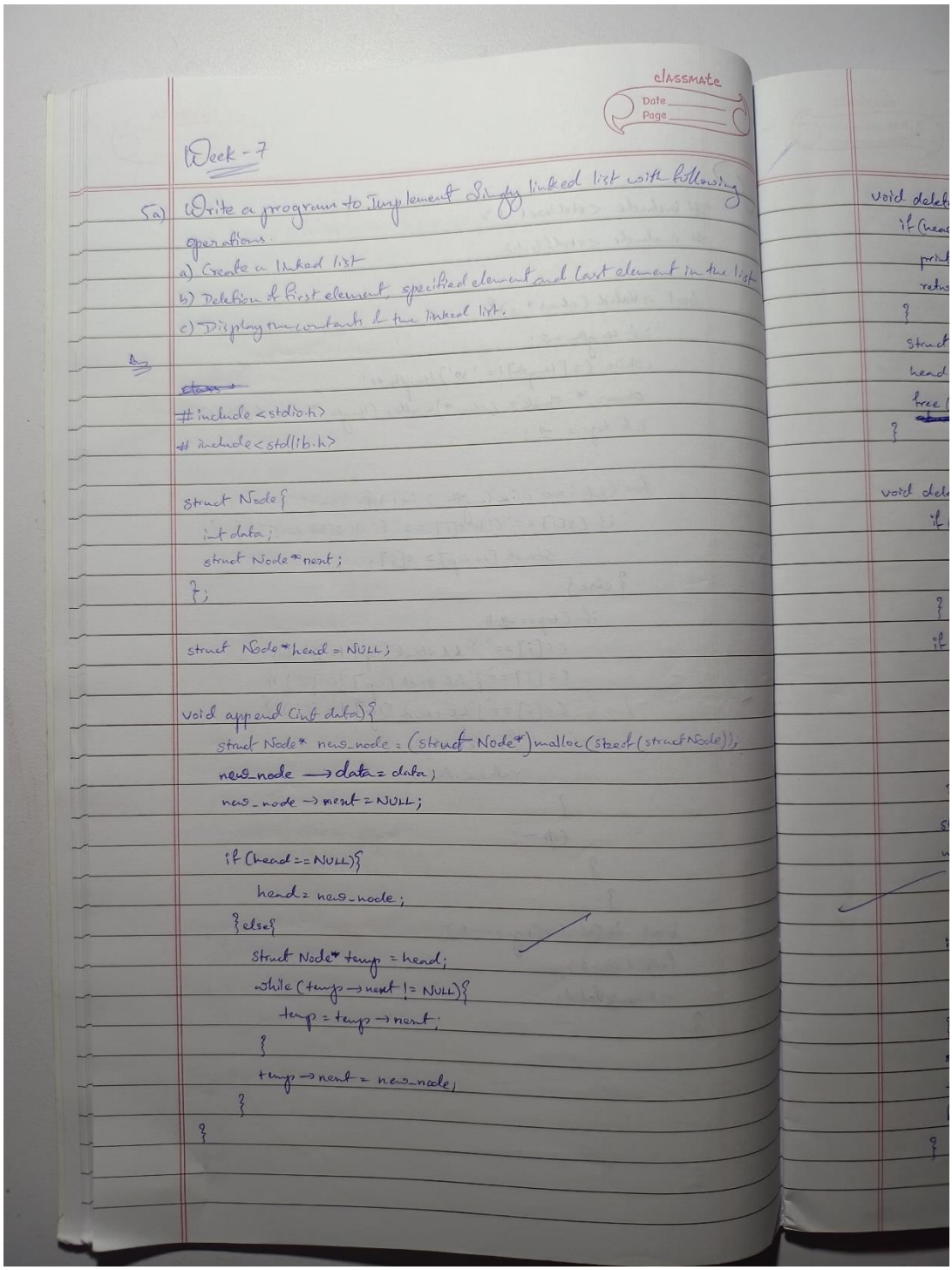
```
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

Output:



5. 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

Observation:



Week - 7

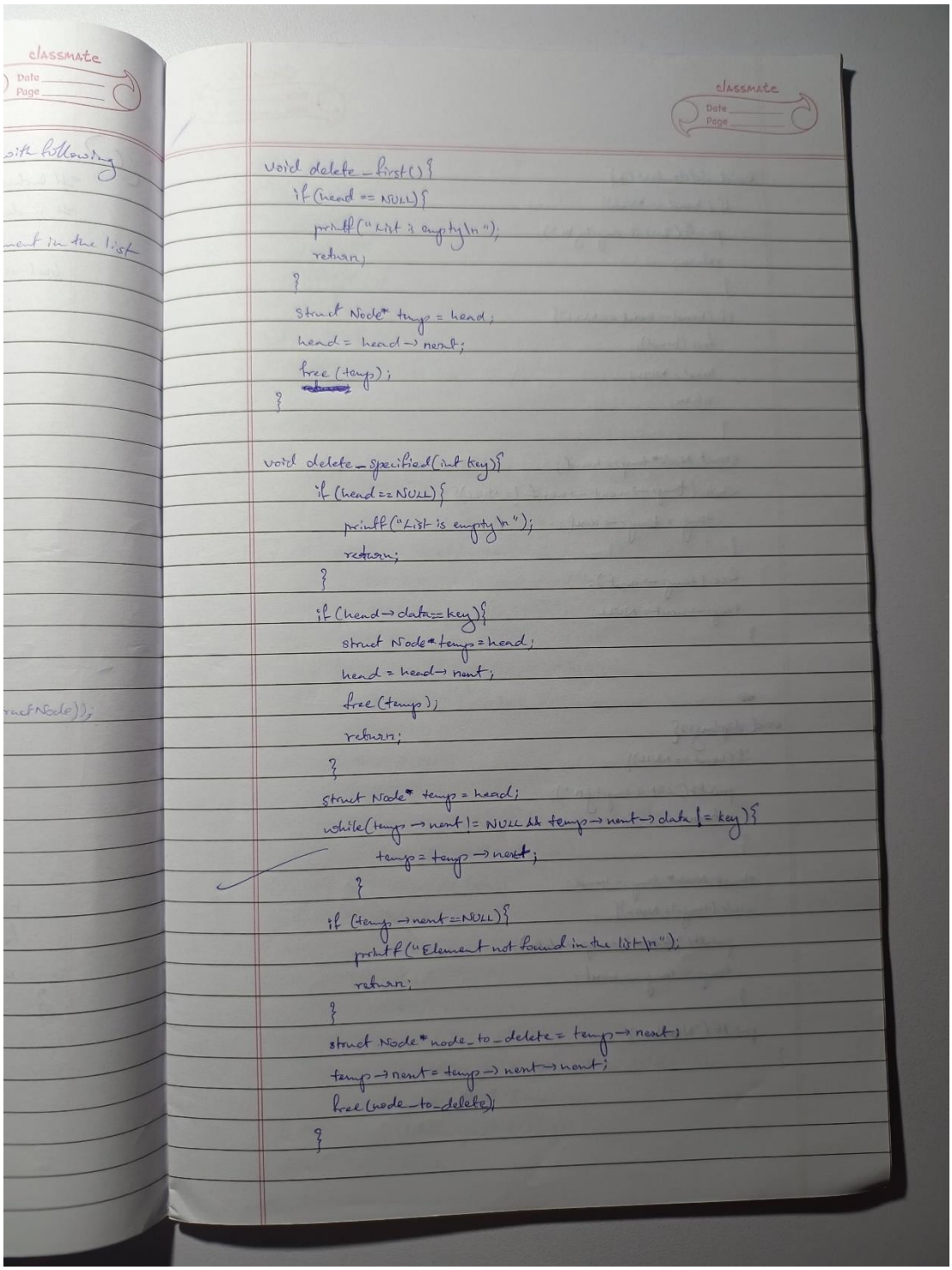
5a) Write a program to Implement Singly linked list with following operations.

a) Create a linked list

b) Deletion of first element, specified element and last element in the list

c) Display the contents of the linked list.

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

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

struct Node *head = NULL;

void append(int data){
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node -> data = data;
  new_node -> next = NULL;

  if(head == NULL){
    head = new_node;
  }else{
    struct Node* temp = head;
    while(temp -> next != NULL){
      temp = temp -> next;
    }
    temp -> next = new_node;
  }
}
```

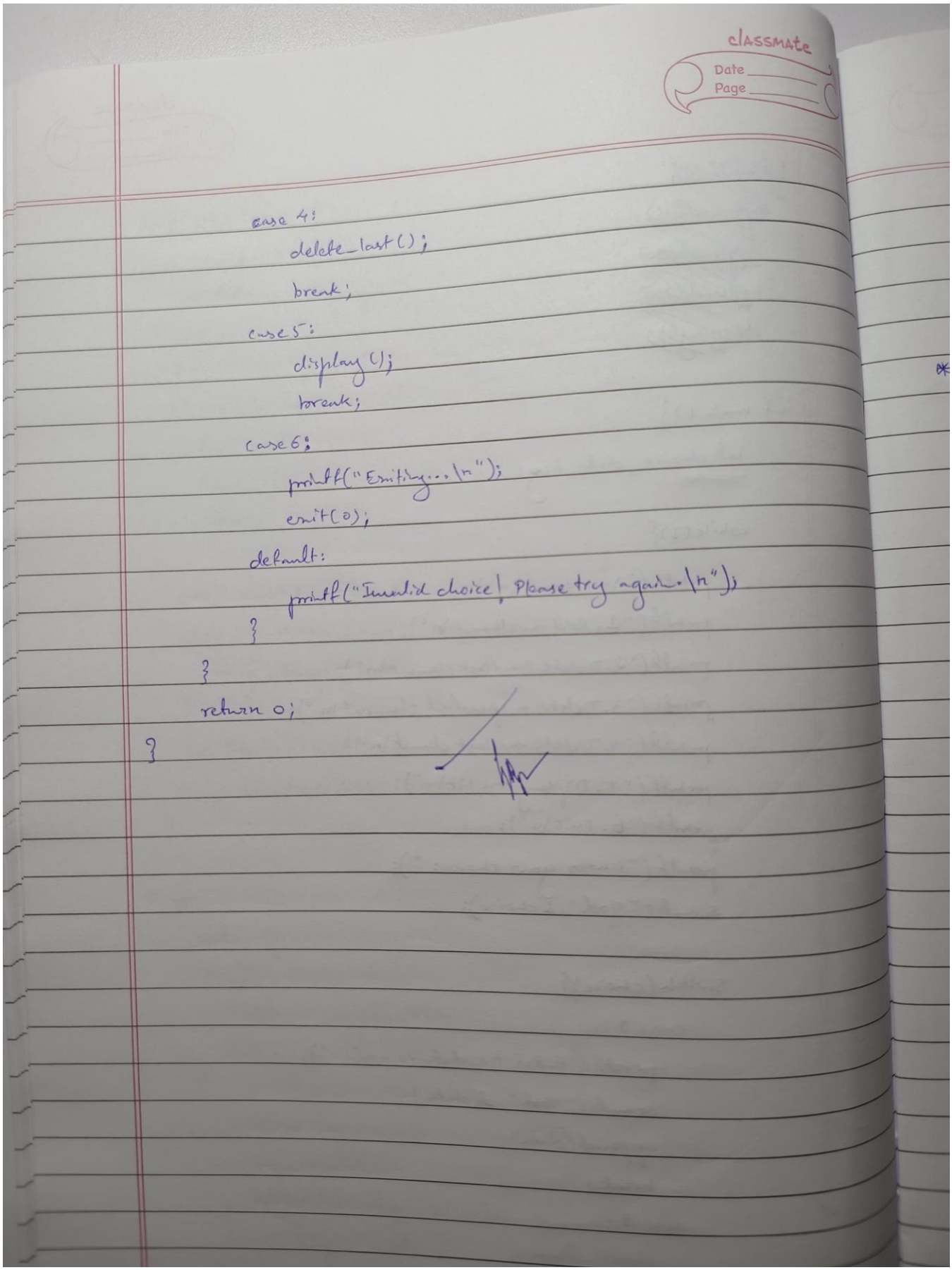


```
void delete_first(){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head -> next;
    free(temp);
}

void delete_specified(int key){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }
    if(head -> data == key){
        struct Node* temp = head;
        head = head -> next;
        free(temp);
        return;
    }
    struct Node* temp = head;
    while(temp -> next != NULL && temp -> next -> data != key){
        temp = temp -> next;
    }
    if(temp -> next == NULL){
        printf("Element not found in the list\n");
        return;
    }
    struct Node* node_to_delete = temp -> next;
    temp -> next = temp -> next -> next;
    free(node_to_delete);
}
```

```
void delete_last(){
  if (head == NULL){
   printf("List is empty\n");
   return;
  }
  if (head -> next == NULL){
    free (head);
    head = NULL;
    return;
   }
  struct Node* temp = head;
  while (temp -> next -> next != NULL){
     temp = temp -> next;
  }
  free (temp -> next);
  temp -> next = NULL;
}


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

~~int main(){~~
~~append(10);~~
~~append(20);~~
~~append(30);~~
~~display();~~

```
int main(){
    int choice, data, key;

    while(1){
        printf("\n Menu:\n");
        printf("1. Add an element\n");
        printf("2. Delete the first element\n");
        printf("3. Delete a specified element\n");
        printf("4. Delete the last element\n");
        printf("5. Display the list\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%od", &choice);

        switch(choice){
            case 1:
                printf("Enter the data to add: ");
                scanf("%od", &data);
                append(data);
                break;
            case 2:
                delete_first();
                break;
            case 3:
                printf("Enter the element to delete: ");
                scanf("%od", &key);
                delete_specified(key);
                break;
```



```
        case 4:
            delete_last();
            break;
        case 5:
            display();
            break;
        case 6:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output :-

Menu:
1. Add an element
2. Delete the first element
3. Delete a specified element
4. Delete the last element
5. Display the list
6. Exit

Enter your choice: 1
Enter the data to add: 12

*1
Enter the data to add: 13

*1
Enter the data to add: 14

*1
Enter the data to add: 15

*1
Enter the data to add: 16

*1
Enter the data to add: 17

*5
12 → 13 → 14 → 15 → 16 → 17 → NULL

*2

*5
13 → 14 → 15 → 16 → 17 → NULL

*3
Enter the element to delete: 15

*5
13 → 14 → 16 → 17 → NULL

*4

*5
13 → 14 → 16 → NULL

*2

*2

*2

*2
List is empty

*6
Exiting...

Code:

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

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

struct Node* head = NULL;

void createLinkedList(int n) {
    struct Node *newNode, *temp;
    int data, i;

    for (i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);

        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = NULL;

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

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

void deleteFirst() {
    if (head == NULL) {
        printf("The list is empty. No node to delete.\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("First node deleted.\n");
}

void deleteSpecified(int key) {
    if (head == NULL) {
        printf("The list is empty. No node to delete.\n");
        return;
    }

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

    if (temp != NULL && temp->data == key) {
        head = temp->next;
        free(temp);
        printf("Node with data %d deleted.\n", key);
```

```
        return;
    }

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

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

    prev->next = temp->next;
    free(temp);
    printf("Node with data %d deleted.\n", key);
}

void deleteLast() {
    if (head == NULL) {
        printf("The list is empty. No node to delete.\n");
        return;
    }

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

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

```
        printf("Last node deleted.\n");
        return;
    }

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

    prev->next = NULL;
    free(temp);
    printf("Last node deleted.\n");
}

void displayList() {
    struct Node* temp = head;

    if (head == NULL) {
        printf("The list is empty.\n");
    } else {
        printf("Linked list contents: ");
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
```

```
int main() {
    int choice, data, n;

    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Create Linked List\n2. Delete First\n3. Delete Specified\n4. Delete Last\n5. Display List\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of nodes: ");
                scanf("%d", &n);
                createLinkedList(n);
                break;
            case 2:
                deleteFirst();
                break;
            case 3:
                printf("Enter the value to delete: ");
                scanf("%d", &data);
                deleteSpecified(data);
                break;
            case 4:
                deleteLast();
                break;
            case 5:
                displayList();
                break;
```

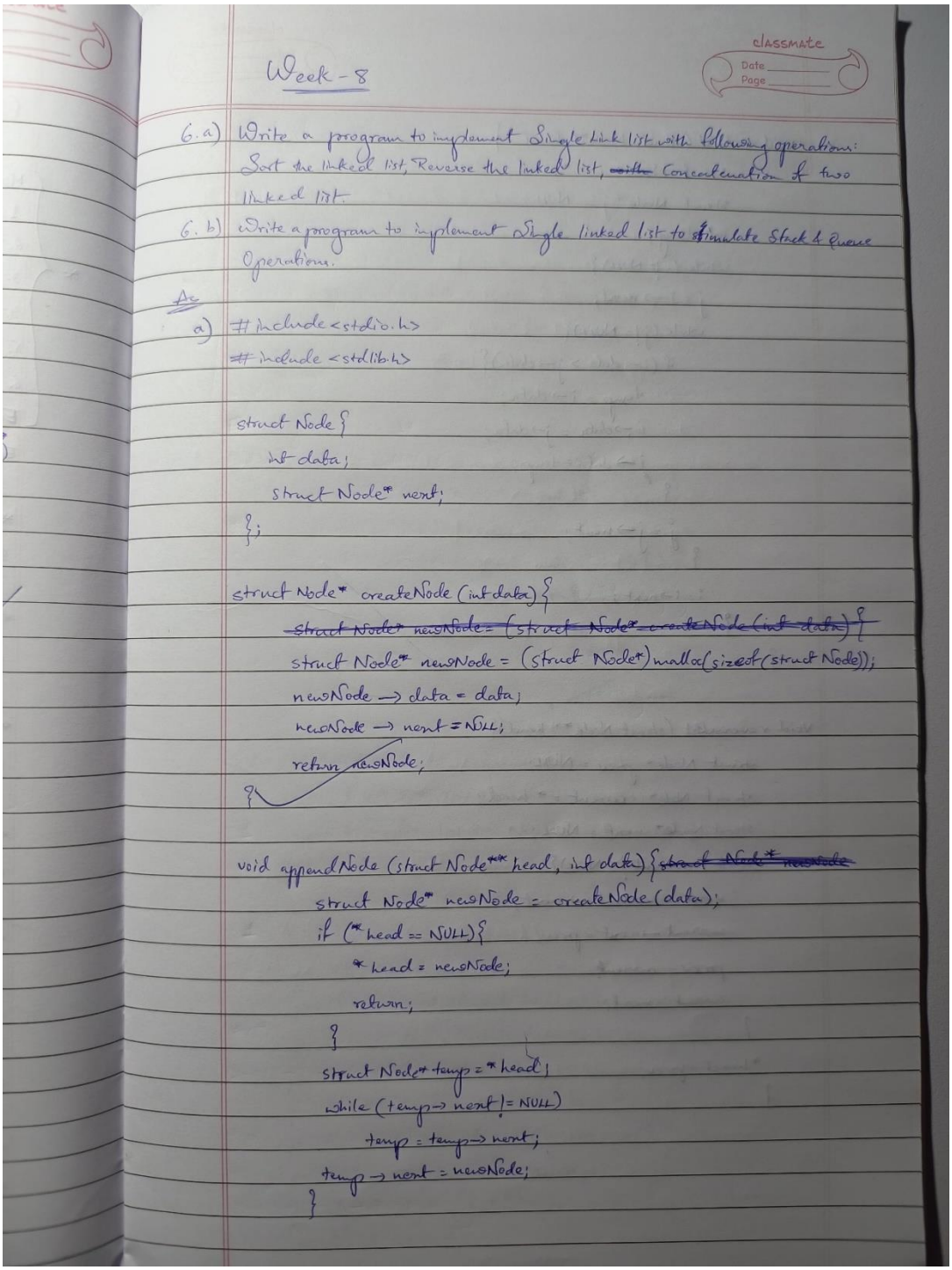
```
        case 6:
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 1
Enter the number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 10 -> 20 -> 30 -> NULL

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 2
First node deleted.

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 20 -> 30 -> NULL

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 3
Enter the value to delete: 30
Node with data 30 deleted.

Singly Linked List Operations:
1. Create Linked List
2. Delete First
3. Delete Specified
4. Delete Last
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 20 -> NULL
```

6. WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists, Implement Single Link List to simulate Stack & Queue Operations.

Observation:



Week - 8

6.a) Write a program to implement Single Link list with following operations: Sort the linked list, Reverse the linked list, ~~with~~ Concatenation of two linked list

6.b) Write a program to implement Single linked list to simulate Stack & Queue Operations.

As

a)

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

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

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

void appendNode(struct Node** head, int data){
    struct Node* newNode = createNode(data);
    if (*head == NULL){
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp -> next != NULL)
        temp = temp -> next;
    temp -> next = newNode;
}
```

```
Void sortList(struct Node** head ){
    struct Node* i = *head;
    struct Node *j = NULL;
    int temp;
    while (i != NULL){
      j = i -> next;
      while (j != NULL){
        if (i -> data > j -> data){
          temp = i -> data;
          i -> data = j -> data;
          j -> data = temp;
        }
        j = j -> next;
      }
      i = i -> next;
    }
}
```

```
Void reverseList (struct Node** head){
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;
    while (current != NULL){
      next = current -> next;
      current -> next = prev;
      prev = current;
      current = next;
    }
    *head = prev;
}
```

```
void concatenateLists(struct Node** head1, struct Node** head2){
    if (*head1 == NULL){
        *head1 = *head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = *head2;
}

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

int main(){
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice, n, data;

    while (1) {
    printf("\nMenu:\n");
    printf("1. Add elements to List1\n");
    printf("2. Add elements to List2\n");
    printf("3. Sort List 1\n");
    printf("4. Reverse List 1\n");
    printf("5. Concatenate List 2 to List 1\n");
    printf("6. Display List 1\n");
    printf("7. Display List 2\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
```

```
scanf("%d", &choice);
switch (choice){
case 1: printf("Enter the number of element to add to List 1 : ");
        scanf("%d", &n);
        printf("Enter elements: ");
        for (int i=0; i<n; i++){
            scanf("%d", &data);
            appendNode(&list1, data);
        }
        break;
case 2: printf("Enter the number of element to add to List 2 : ");
        scanf("%d", &n);
        printf("Enter elements:");
        for (int i=0; i<n; i++){
            scanf("%d", &data);
            appendNode(&list2, data);
        }
        break;
case 3: sortList(&list1);
        printf("List 1 sorted.\n");
        break;
case 4: reverseList(&list1);
        printf("List 1 reversed.\n");
        break;
case 5: concatenateLists(&list1, &list2);
        printf("List 2 concatenated to List 1.\n");
        break;
case 6: printf("List1: ");
        displayList(list1);
        break;
case 7: printf("List2:");
        displayList(list2);
        break;
case 8: exit(0);
```

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

Output :-

```
Menu
1. Add element
2. Add element
3. Sort List
4. Reverse
5. Concatenate
6. Display List
7. Display List
Enter your
Enter the n
Enter the
*6
List1:
*2
Enter the
Enter the
*7
List2:
*4
List1 rev
```

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

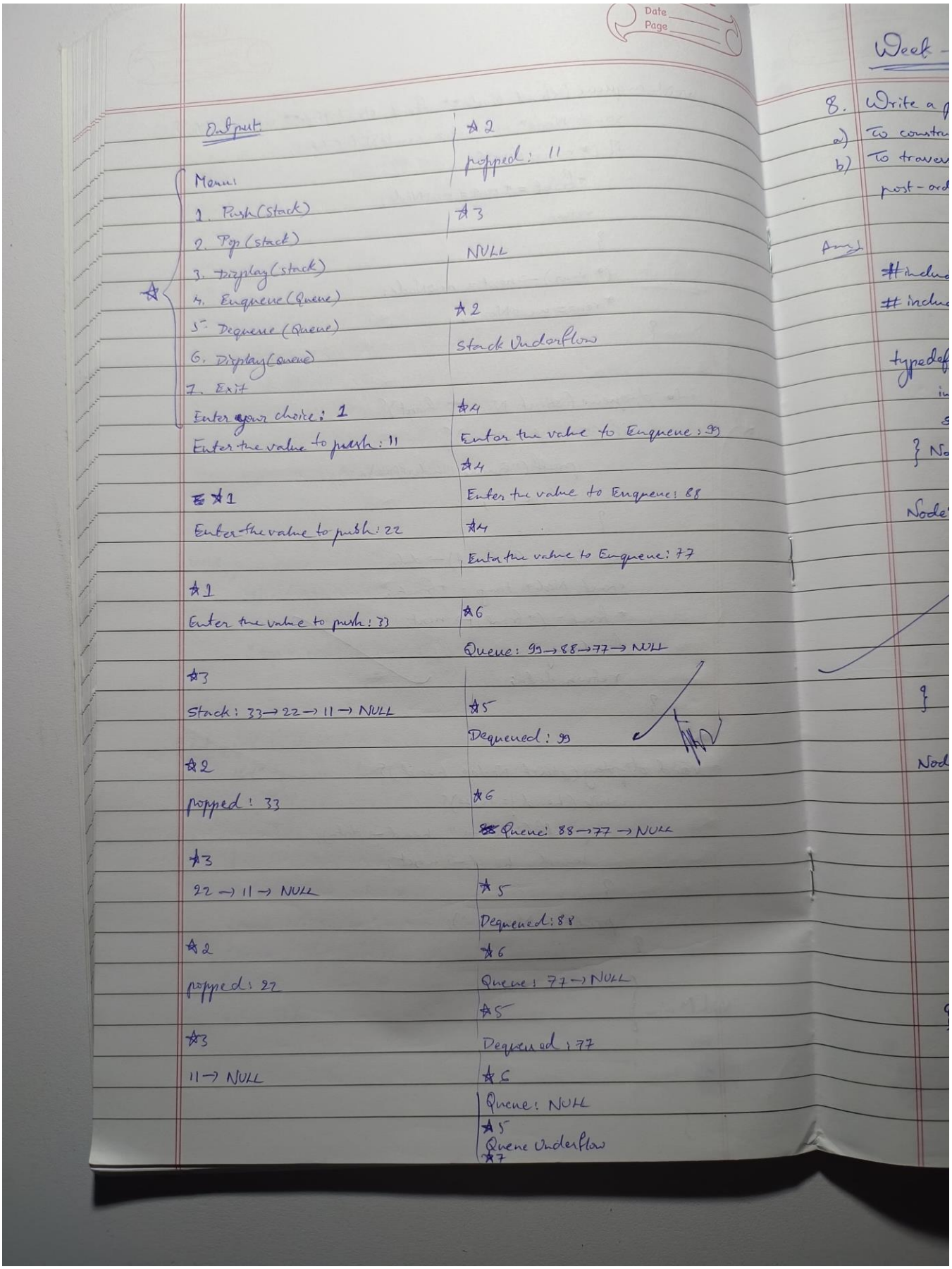
Output:-

```
Menu
1. Add element to List1
2. Add element to List2
3. Sort List1
4. Reverse List1
5. Concatenate List2 to List1
6. Display List1
7. Display List2

Enter your choice: 1
Enter the number of elements to add to List1: 3
Enter the elements: 11 22 33

*6
List1: 11 -> 22 -> 33 -> NULL

*2
Enter the number of elements to add to List2: 3
Enter the elements: 44 55 66

*7
List2: 44 -> 55 -> 66 -> NULL

*4
List1 reversed.

*6
List1: 33 -> 22 -> 11 -> NULL

*2
List1 sorted

*6
List1: 11 -> 22 -> 33 -> NULL

*5
List2 has been concatenated to List1

*6
List1: 11 -> 22 -> 33 -> 44 -> 55 -> 66 -> NULL

*7
List2: 44 -> 55 -> 66 -> NULL

*8
```

5)

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

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

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

void push(struct Node** top, int data){
    struct Node* newNode = createNode(data);
    newNode -> next = *top;
    *top = newNode;
}

int pop(struct Node** top){
    if(*top = NULL){
        printf("stack Underflow\n");
        return -1;
    }
    int data = (*top) -> data;
    struct Node* temp = *top;
    *top = (*top) -> next;
    free(temp);
    return data;
}
```

```
void enqueue(struct Node** front, struct Node** rear, int data){
    struct Node* newNode = createNode(data);
    if (*rear == NULL){
        *front = *rear = newNode;
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(struct Node** front){
    if (*front == NULL){
        printf("Queue Underflow\n");
        return -1;
    }
    int data = (*front)->data;
    struct Node* temp = *front;
    *front = (*front)->next;
    free(temp);
    return data;
}

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

Void Main{
}
```



Output:

Menu:
1. Push(stack)
2. Pop(stack)
3. Display(stack)
4. Enqueue(Queue)
5. Dequeue(Queue)
6. Display(Queue)
7. Exit

☆

Enter your choice: 1
Enter the value to push: 11

☆1
Enter the value to push: 22

☆1
Enter the value to push: 33

☆3
Stack: 33 → 22 → 11 → NULL

☆2
popped: 33

☆3
22 → 11 → NULL

☆2
popped: 22

☆3
11 → NULL

☆2
popped: 11

☆3
NULL

☆2
Stack Underflow

☆4
Enter the value to Enqueue: 99

☆4
Enter the value to Enqueue: 88

☆4
Enter the value to Enqueue: 77

☆6
Queue: 99 → 88 → 77 → NULL

☆5
Dequeued: 99

☆6
Queue: 88 → 77 → NULL

☆5
Dequeued: 88

☆6
Queue: 77 → NULL

☆5
Dequeued: 77

☆6
Queue: NULL

☆5
Queue Underflow

☆7

Week -

8. Write a
a) To constru
b) To traver
post-ord

Ans:

#includ
#includ

typedef

} No

Node

Nod

Code:

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

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

struct Node* head = NULL;

void insertAtEnd(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp;
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void sortList() {
    struct Node* i;
```

```
    struct Node* j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
    printf("Linked list sorted.\n");
}

void reverseList() {
    struct Node *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    printf("Linked list reversed.\n");
}

void concatenateLists(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) {
        head = head2;
```

```
        return;
    }
    struct Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
    head = head1;
    printf("Linked lists concatenated.\n");
}

void displayList() {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("The list is empty.\n");
    } else {
        printf("Linked list contents: ");
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}

int main() {
    int choice, data, n, i;
    struct Node *list1 = NULL, *list2 = NULL;
```

```
    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at End\n2. Sort List\n3. Reverse List\n4. Concatenate Two Lists\n5.
Display List\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
                insertAtEnd(data);
                break;
            case 2:
                sortList();
                break;
            case 3:
                reverseList();
                break;
            case 4:
                printf("Enter number of nodes for first list: ");
                scanf("%d", &n);
                for (i = 0; i < n; i++) {
                    printf("Enter data for node %d of list1: ", i + 1);
                    scanf("%d", &data);
                    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
                    newNode->data = data;
                    newNode->next = list1;
                    list1 = newNode;
                }
```

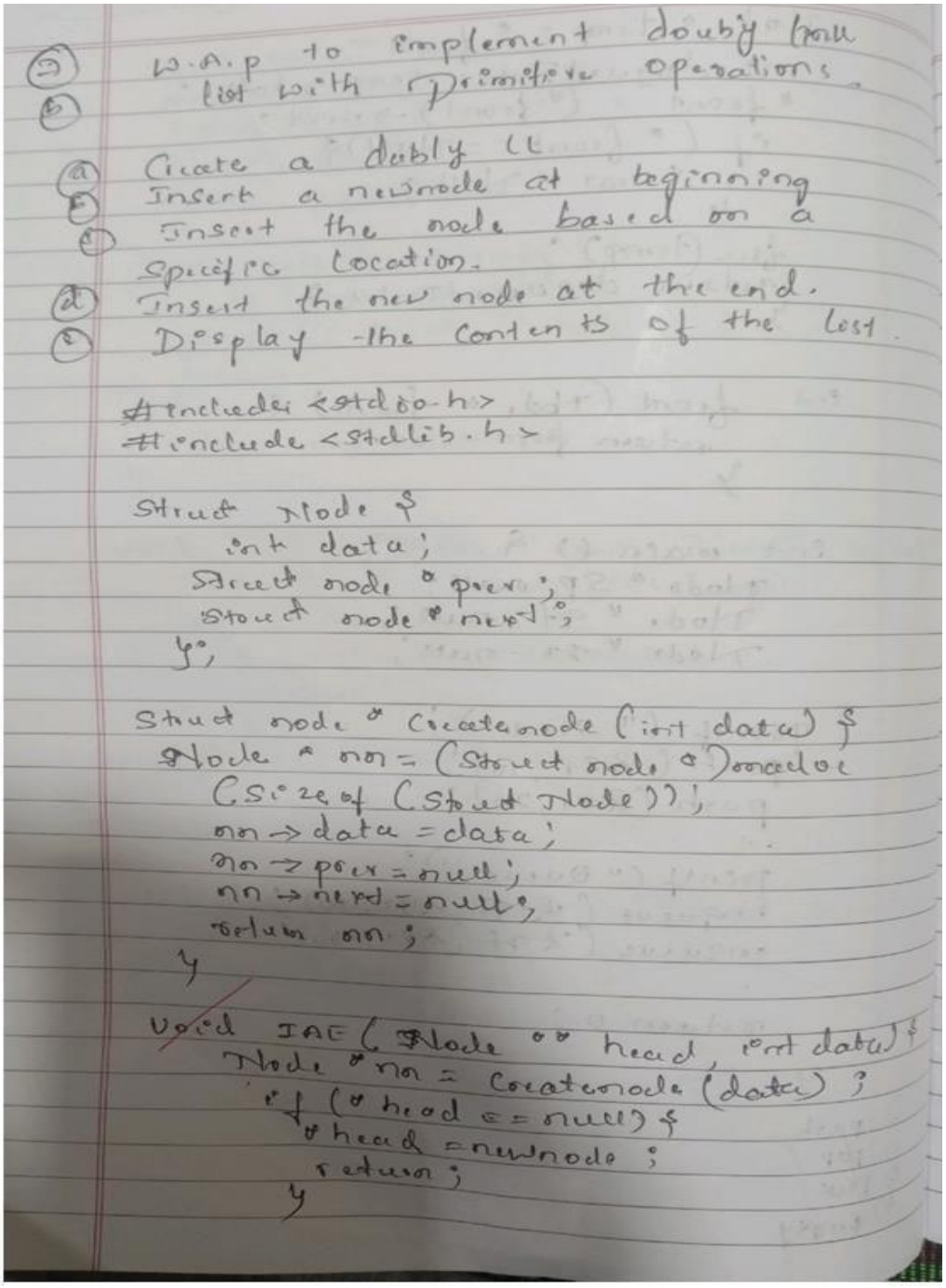
```
            printf("Enter number of nodes for second list: ");
            scanf("%d", &n);
            for (i = 0; i < n; i++) {
                printf("Enter data for node %d of list2: ", i + 1);
                scanf("%d", &data);
                struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
                newNode->data = data;
                newNode->next = list2;
                list2 = newNode;
            }
            concatenateLists(list1, list2);
            break;
        case 5:
            displayList();
            break;
        case 6:
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 1
Enter data to insert at end: 10

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 1
Enter data to insert at end: 20

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 10 -> 20 -> NULL

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 2
Linked list sorted.

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 5
Linked list contents: 10 -> 20 -> NULL

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
5. Display List
6. Exit
Enter your choice: 3
Linked list reversed.

Singly Linked List Operations:
1. Insert at End
2. Sort List
3. Reverse List
4. Concatenate Two Lists
Linked list contents: 5 -> 15 -> 25 -> 30 -> NULL
```

7. 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

Observation:



W.A.P to implement doubly linked list with Primitive operations.

a. Create a doubly LL
b. Insert a newnode at beginning
c. Insert the node based on a specific location.
d. Insert the new node at the end.
e. Display the contents of the list.

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

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

struct node * Createnode (int data) {
  Node * nn = (struct node *)malloc
    (sizeof (struct Node));
  nn -> data = data;
  nn -> prev = null;
  nn -> next = null;
  return nn;
}

void IAE ( Node ** head, int data) {
  Node * nn = Createnode (data);
  if (* head == null) {
    * head = newnode;
    return;
  }
```

```
struct Node * temp = * head;
while (temp -> next != null) temp = temp -> next;
temp -> next = newnode;
nn -> prev = temp;
}

void IAL (Node * head, int data, int la){
  struct Node * nn = CreateNode(data);
  if (loc == 0) {
    IAB (head, data);
    return;
  }
}

void displaylist (struct Node * head) {
  struct Node * temp = head;
  while (temp != null) {
  printf ("%d ", temp -> data);
    temp = temp -> next;
  }
}

int main () {
  Node * head = null;

  IAB (&head, 10);
  IAB (&head, 20);
  IAE (&head, 30);
  IAL (&head, 25, 2);

  displaylist (head);
  return 0;
}
```

O/P

1) IAB
2) IAE
3) IAL
4) Display
5 Exit

1) 5
2) 6
3) 7

2) 8
3) 9
2) 12

4) 5 6 7 8 12 9 8

5) Editing

3) IAL
Index 3

5 6 7 12 3 9 8

5) Exit

Code:

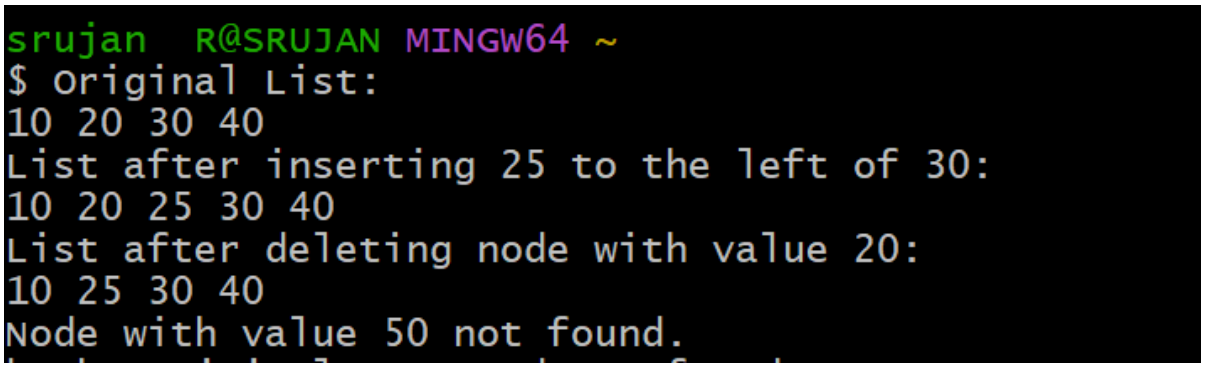
```
#include <stdio.h>
#include <stdlib.h>

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

struct DoublyLinkedList {
    struct Node* head;
};

void createList(struct DoublyLinkedList* dll, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = newNode->next = NULL;

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

```
}

void insertLeft(struct DoublyLinkedList* dll, int newData, int existingData) {
    struct Node* temp = dll->head;
    while (temp != NULL && temp->data != existingData) {
        temp = temp->next;
    }

    if (temp != NULL) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = newData;
        newNode->next = temp;
        newNode->prev = temp->prev;

        if (temp->prev != NULL) {
            temp->prev->next = newNode;
        } else {
            dll->head = newNode;
        }
        temp->prev = newNode;
    } else {
        printf("Node with data %d not found.\n", existingData);
    }
}

void deleteNode(struct DoublyLinkedList* dll, int value) {
    struct Node* temp = dll->head;

    while (temp != NULL && temp->data != value) {
```

```
        temp = temp->next;
    }

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

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

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

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

void display(struct DoublyLinkedList* dll) {
    if (dll->head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = dll->head;
    while (temp != NULL) {
```

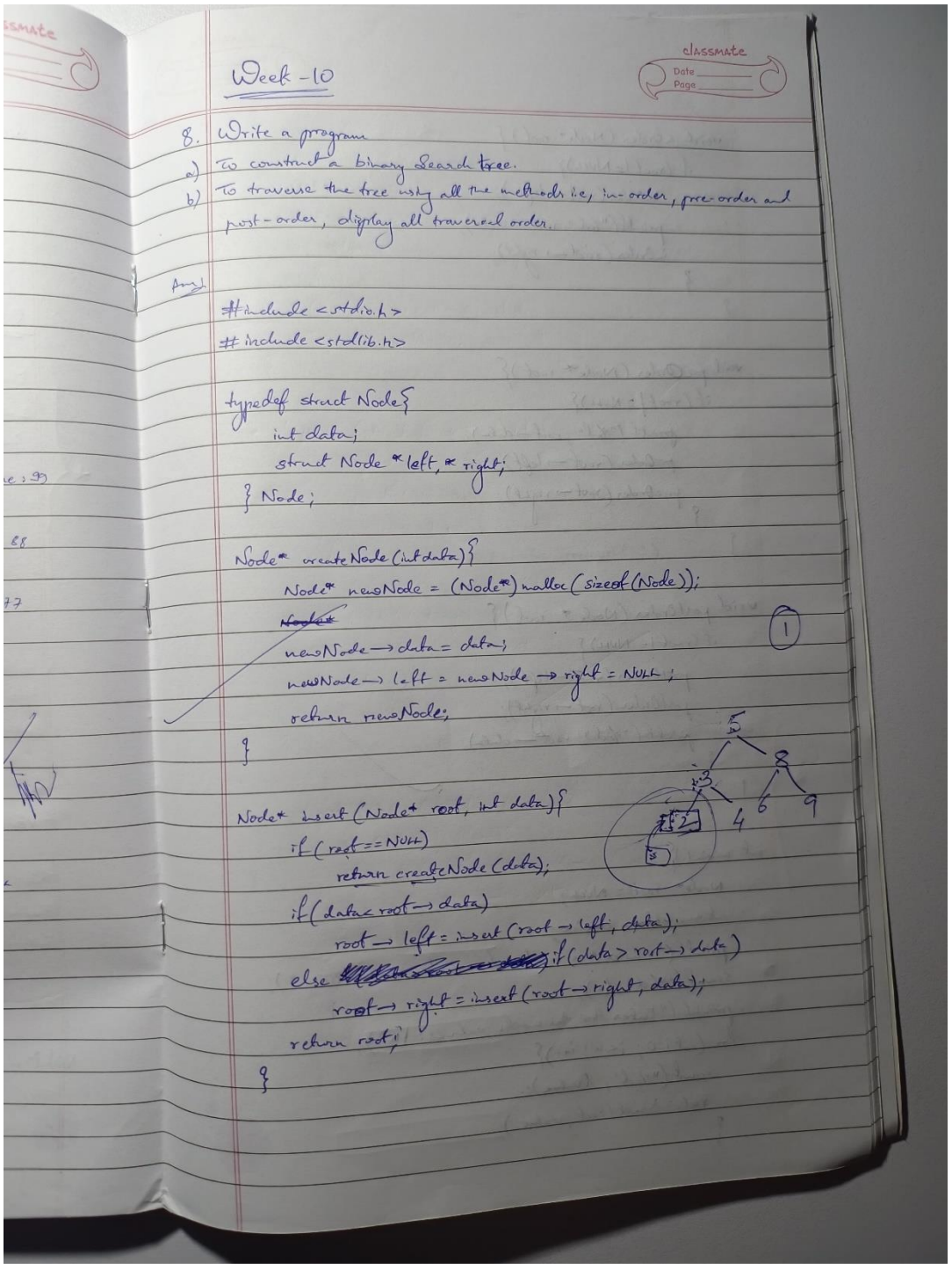
```
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct DoublyLinkedList dll;
    dll.head = NULL;

    createList(&dll, 10);
    createList(&dll, 20);
    createList(&dll, 30);
    createList(&dll, 40);

    printf("Original List:\n");
    display(&dll);

    insertLeft(&dll, 25, 30);
    printf("List after inserting 25 to the left of 30:\n");
    display(&dll);

    deleteNode(&dll, 20);
    printf("List after deleting node with value 20:\n");
    display(&dll);

    deleteNode(&dll, 50);
    return 0;
}
```

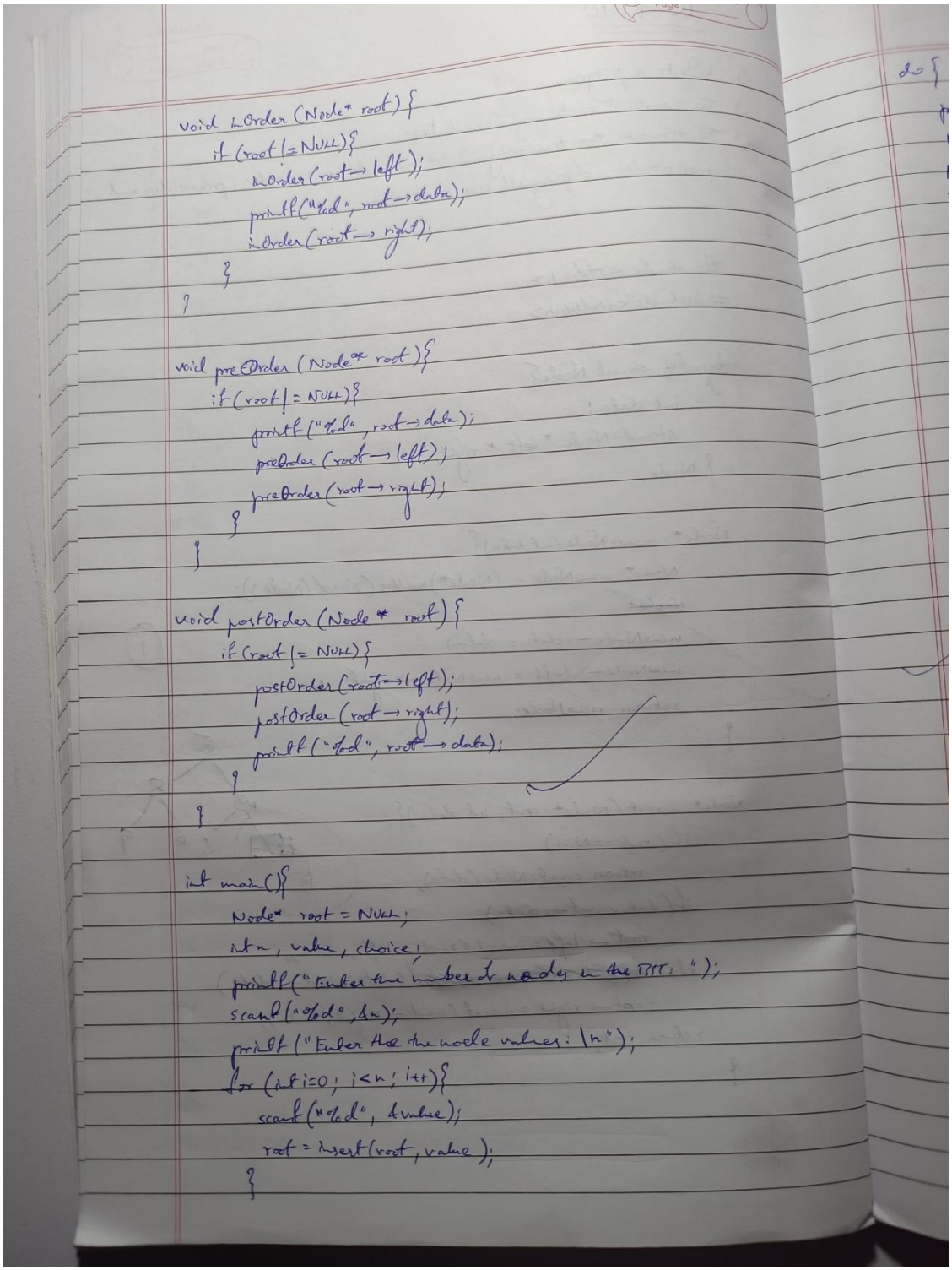
Output:


srujan R@SRUJAN MINGW64 ~
$ Original List:
10 20 30 40
List after inserting 25 to the left of 30:
10 20 25 30 40
List after deleting node with value 20:
10 25 30 40
Node with value 50 not found.

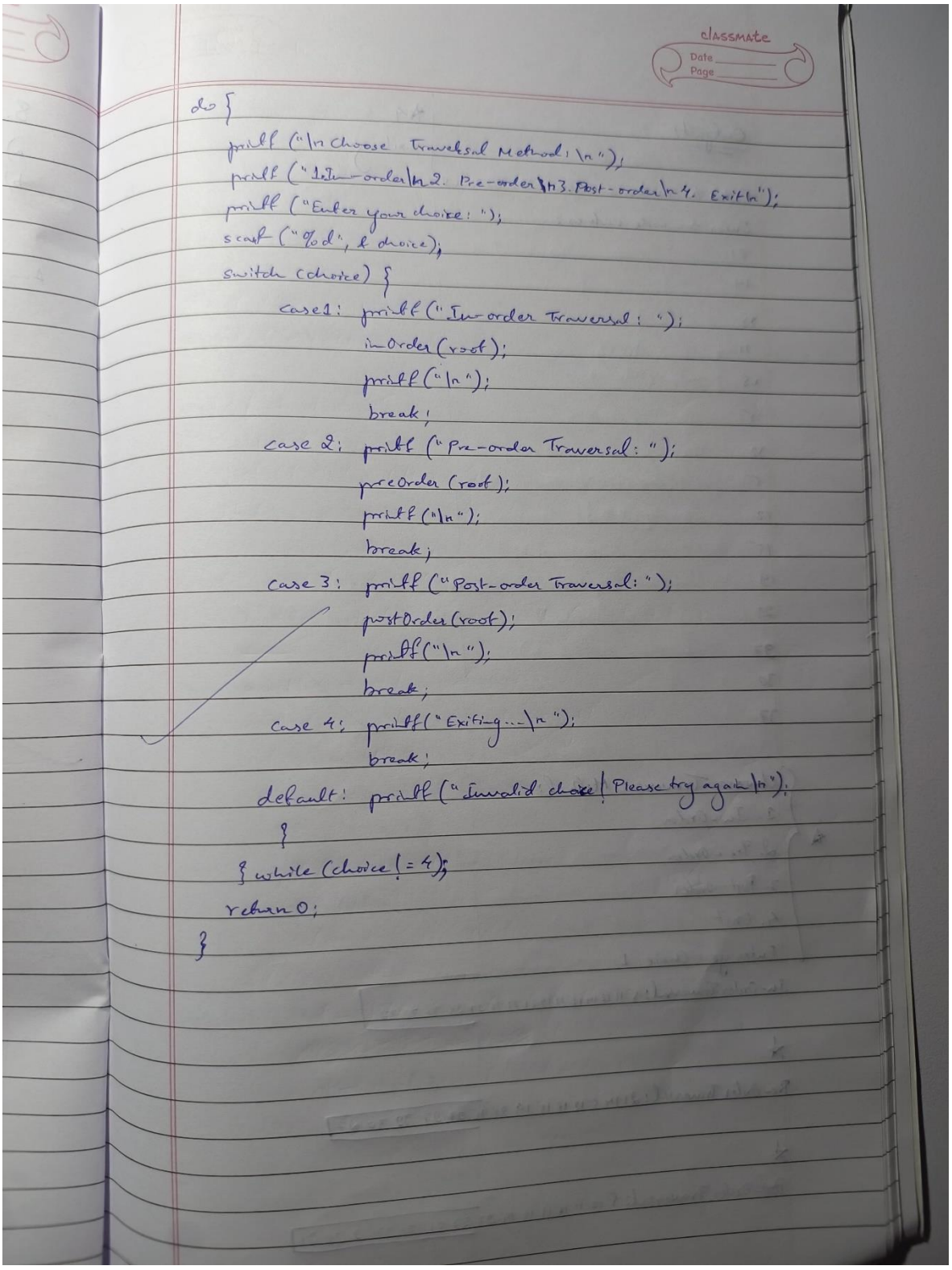
8. Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in order, preorder and post order c) To display the elements in the tree.

Observation:

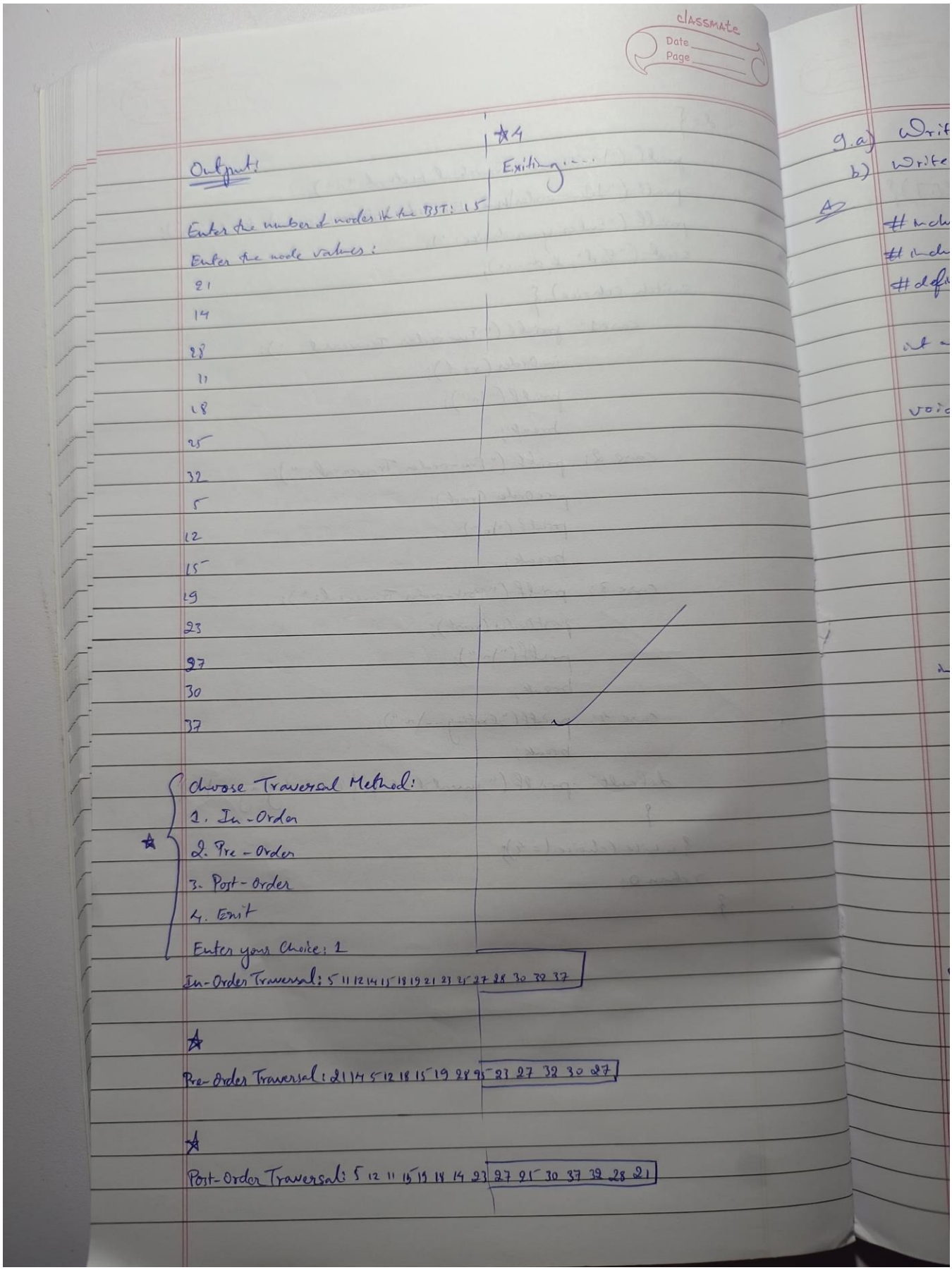




```
void inOrder (Node* root) {
    if (root != NULL){
        inOrder (root -> left);
        printf("%d", root -> data);
        inOrder (root -> right);
    }
}

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

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

int main(){
    Node* root = NULL;
    int n, value, choice;
    printf("Enter the number of nodes in the BST: ");
    scanf("%d",&n);
    printf("Enter the the node values: \n");
    for (int i=0; i<n; i++){
        scanf("%d", &value);
        root = Insert(root, value);
    }
```



```
do {
    printf ("\n Choose Traversal Method: \n");
    printf ("1. In-order\n2. Pre-order\n3. Post-order\n4. Exit\n");
    printf ("Enter your choice: ");
    scanf ("%d", &choice);
    switch (choice) {
        case 1: printf ("In-order Traversal: ");
                inOrder (root);
                printf ("\n");
                break;
        case 2: printf ("Pre-order Traversal: ");
                preOrder (root);
                printf ("\n");
                break;
        case 3: printf ("Post-order Traversal: ");
                postOrder (root);
                printf ("\n");
                break;
        case 4: printf ("Exiting...\n");
                break;
        default: printf ("Invalid choice! Please try again\n");
    }
} while (choice != 4);
return 0;
}
```



Output:

Enter the number of nodes in the BST: 15
Enter the node values:
21
14
28
11
18
25
32
5
12
15
19
23
27
30
37

Choose Traversal Method:
1. In-Order
2. Pre-Order
3. Post-Order
4. Exit
Enter your choice: 1
In-Order Traversal: 5 11 12 14 15 18 19 21 23 25 27 28 30 32 37

Pre-Order Traversal: 21 14 5 12 11 15 18 19 28 25 23 27 32 30 37

Post-Order Traversal: 5 12 11 15 19 18 14 23 27 25 30 37 32 28 21

4
Exiting....

9.a) Writ
b) Write

Code:

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

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

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

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}
```

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

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

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

void display(struct Node* root) {
    if (root != NULL) {
        printf("Inorder: ");
        inorder(root);
        printf("\n");
```

```
        printf("Preorder: ");
        preorder(root);
        printf("\n");

        printf("Postorder: ");
        postorder(root);
        printf("\n");
    }
}

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

    do {
        printf("Binary Search Tree Operations:\n");
        printf("1. Insert Element\n");
        printf("2. Display Elements\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            case 2:
                if (root == NULL) {
                    printf("Tree is empty.\n");
                } else {
                    display(root);
                }
                break;

            case 3:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice, please try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

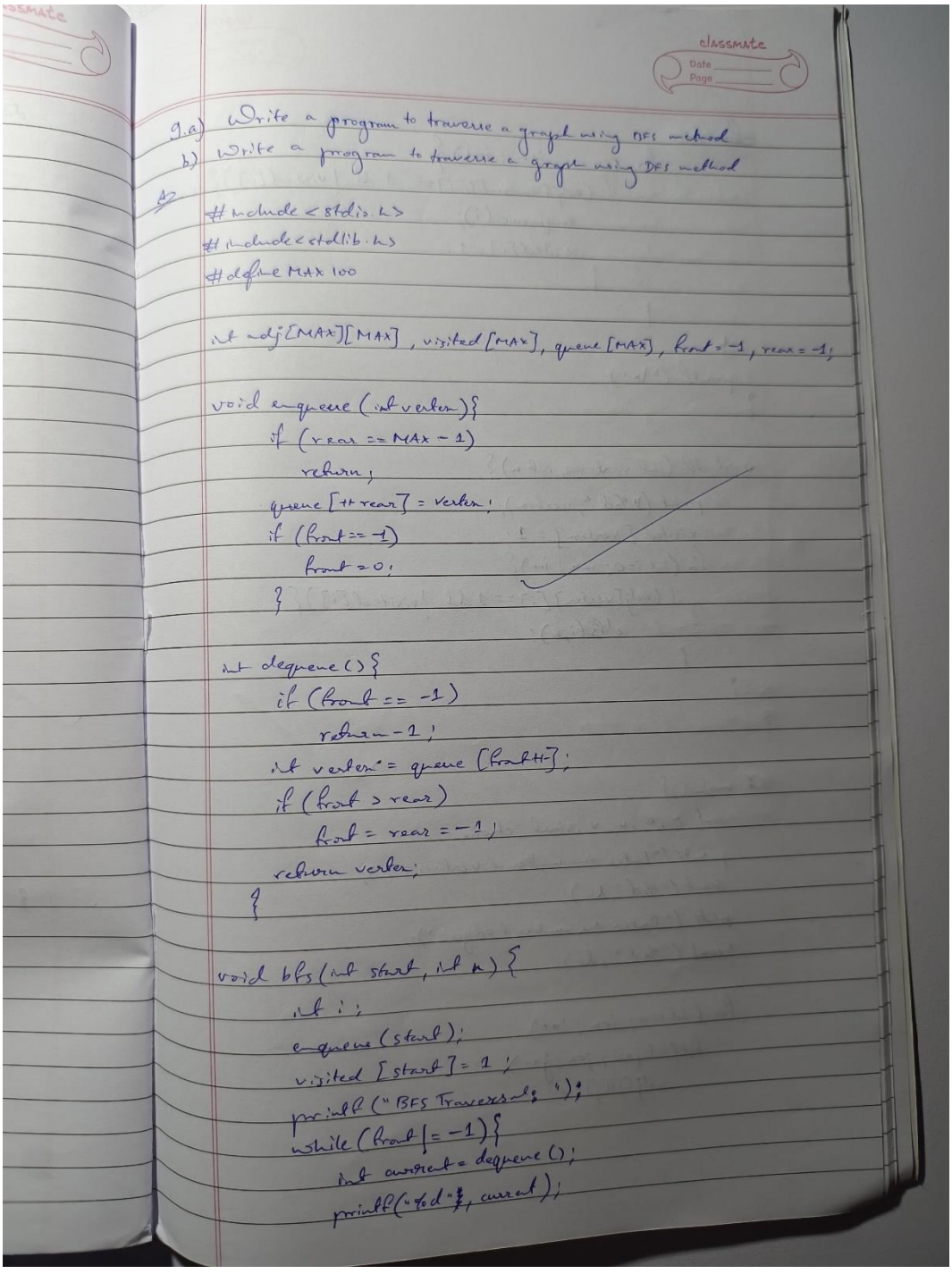
Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 1
Enter value to insert: 50

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 1
Enter value to insert: 30

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 1
Enter value to insert: 70

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 2
Inorder: 30 50 70
Preorder: 50 30 70
Postorder: 30 70 50

Binary Search Tree Operations:
1. Insert Element
2. Display Elements
3. Exit
Enter your choice: 3
Exiting...
```

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

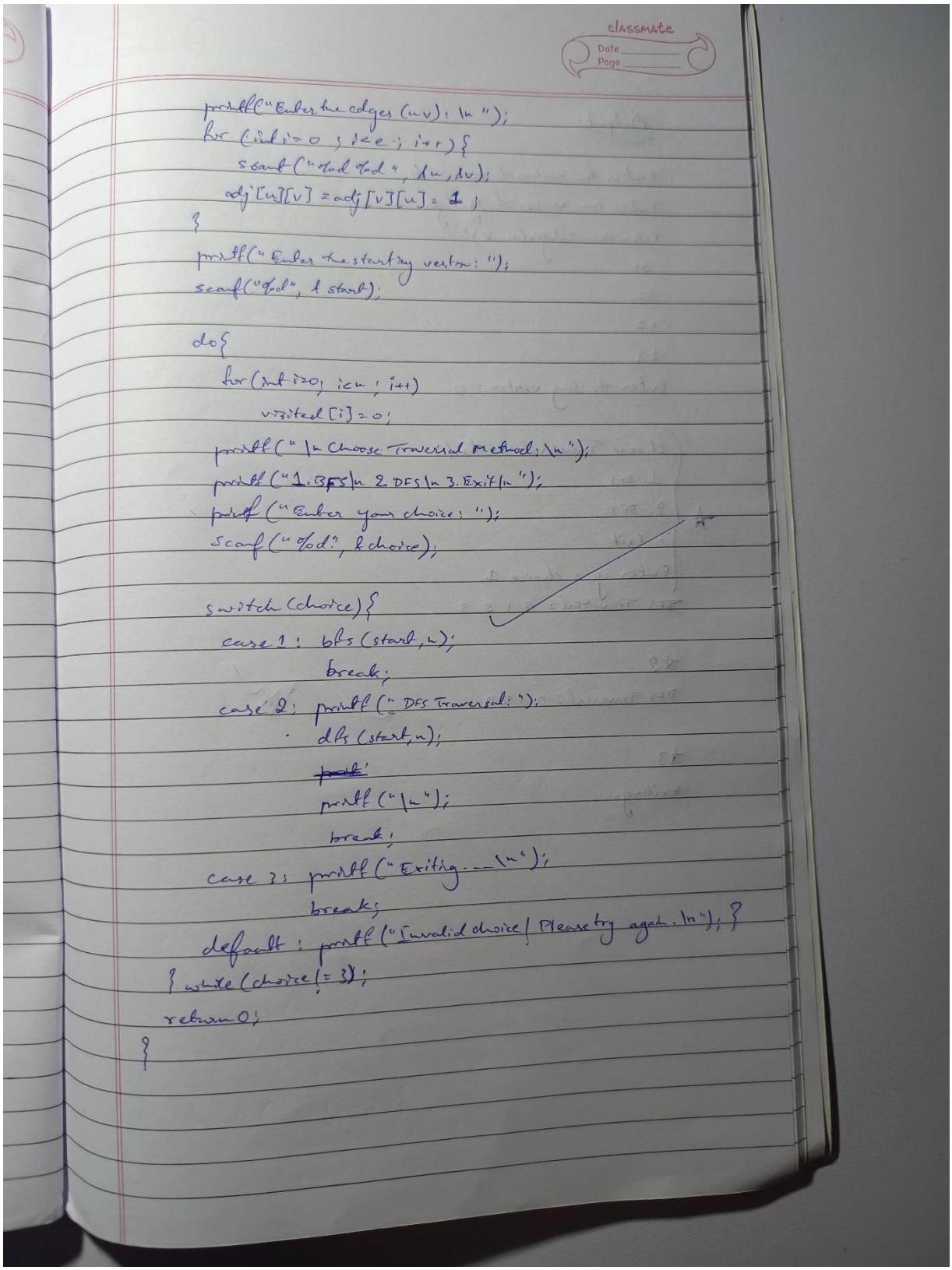
b) Write a program to check whether given graph is connected or not using DFS method.

Observation:



```
9.a) Write a program to traverse a graph using BFS method
b) Write a program to traverse a graph using DFS method

#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int adj[MAX][MAX], visited[MAX], queue[MAX], front = -1, rear = -1;

void enqueue (int vertex){
    if (rear == MAX - 1)
        return;
    queue[++rear] = vertex;
    if (front == -1)
        front = 0;
    }

int dequeue (){
    if (front == -1)
        return -1;
    int vertex = queue[front++];
    if (front > rear)
        front = rear = -1;
    return vertex;
    }

void bfs (int start, int n){
    int i;
    enqueue (start);
    visited [start] = 1;
    printf ("BFS Traversal: ");
    while (front != -1){
        int current = dequeue ();
        printf("%d ", current);
```

```
for (i=0; i<n; i++){
    if (adj[current][i]== 1 && !visited[i]){
        enqueue(i);
        visited[i]=1;
    }
}
printf("\n");
}

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

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

    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            adj[i][j]=0;
```



```
printf("Enter the edges (u v): \n");
for (int i=0 ; i<e ; i++){
    scanf("%d %d", &u, &v);
    adj[u][v] = adj[v][u] = 1;
}
printf("Enter the starting vertex: ");
scanf("%d", &start);

do{
    for(int i=0; i<n ; i++)
        visited[i] = 0;
    printf("\n Choose Traversal Method: \n");
    printf("1. BFS \n 2. DFS \n 3. Exit \n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice){
    case 1: bfs(start, n);
            break;
    case 2: printf("DFS Traversal: ");
            dfs(start, n);
            ~~break~~
            printf("\n");
            break;
    case 3: printf("Exiting... \n");
            break;
    default: printf("Invalid choice! Please try again. \n");}
} while(choice != 3);
return 0;
}
```

Outputs:

Enter the number of vertices: 4
Enter the number of edges: 4
Enter the edges (u v):
0 1
0 2
1 2
2 3
Enter starting vertex: 0

Choose Traversal method:
1. BFS
2. DFS
3. Exit
Enter your choice 1
BFS Traversal: 0 1 2 3

☆2
DFS Traversal: 0 1 2 3

☆3
Exiting....

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
struct Queue {
    int items[MAX];
    int front;
    int rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

bool isEmpty(struct Queue* q) {
    return q->front == -1;
}

bool isFull(struct Queue* q) {
    return q->rear == MAX - 1;
}

void enqueue(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1)
```

```
        q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear)
        q->front = q->rear = -1;
    return item;
}

void BFS(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    struct Queue q;
    initQueue(&q);
    enqueue(&q, start);
    visited[start] = 1;

    while (!isEmpty(&q)) {
        int current = dequeue(&q);
        printf("%d ", current);

        for (int i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
```

```
            enqueue(&q, i);
            visited[i] = 1;
        }
    }
}
}

void DFS(int graph[MAX][MAX], int visited[MAX], int vertex, int n) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < n; i++) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            DFS(graph, visited, i, n);
        }
    }
}

int isConnected(int graph[MAX][MAX], int n) {
    int visited[MAX] = {0};
    DFS(graph, visited, 0, n);

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            return 0;  // Not connected
        }
    }
    return 1;  // Connected
```

```
}

int main() {
    int graph[MAX][MAX] = {0};
    int n, edges, u, v;

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

    for (int i = 0; i < edges; i++) {
        printf("Enter edge (u v): ");
        scanf("%d %d", &u, &v);
        graph[u][v] = 1;
        graph[v][u] = 1;  // For undirected graph
    }

    if (isConnected(graph, n)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

Output:

```
srujan  R@SRUJAN MINGW64 ~
$ Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 1 3
Enter edge (u v): 2 4
BFS traversal starting from vertex 0: 0 1 2 3 4
```

```
srujan  R@SRUJAN MINGW64 ~
$ Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 1 3
Enter edge (u v): 2 4
0 1 3 2 4
```