VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

Soham Hathi (1BM23CS335)

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 to 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 using C Lab (23CS3PCDST)"carried out by **Soham Hathi(1BM23CS335),** who is bonafide student of **B.M.S.College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering inComputer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academicrequirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. Jyothi S Nayak Professor &HOD Department of CSE, BMSCE

Index

Github Link:https://github.com/sohamhathi/1BM23CS335DS.git

Program 1

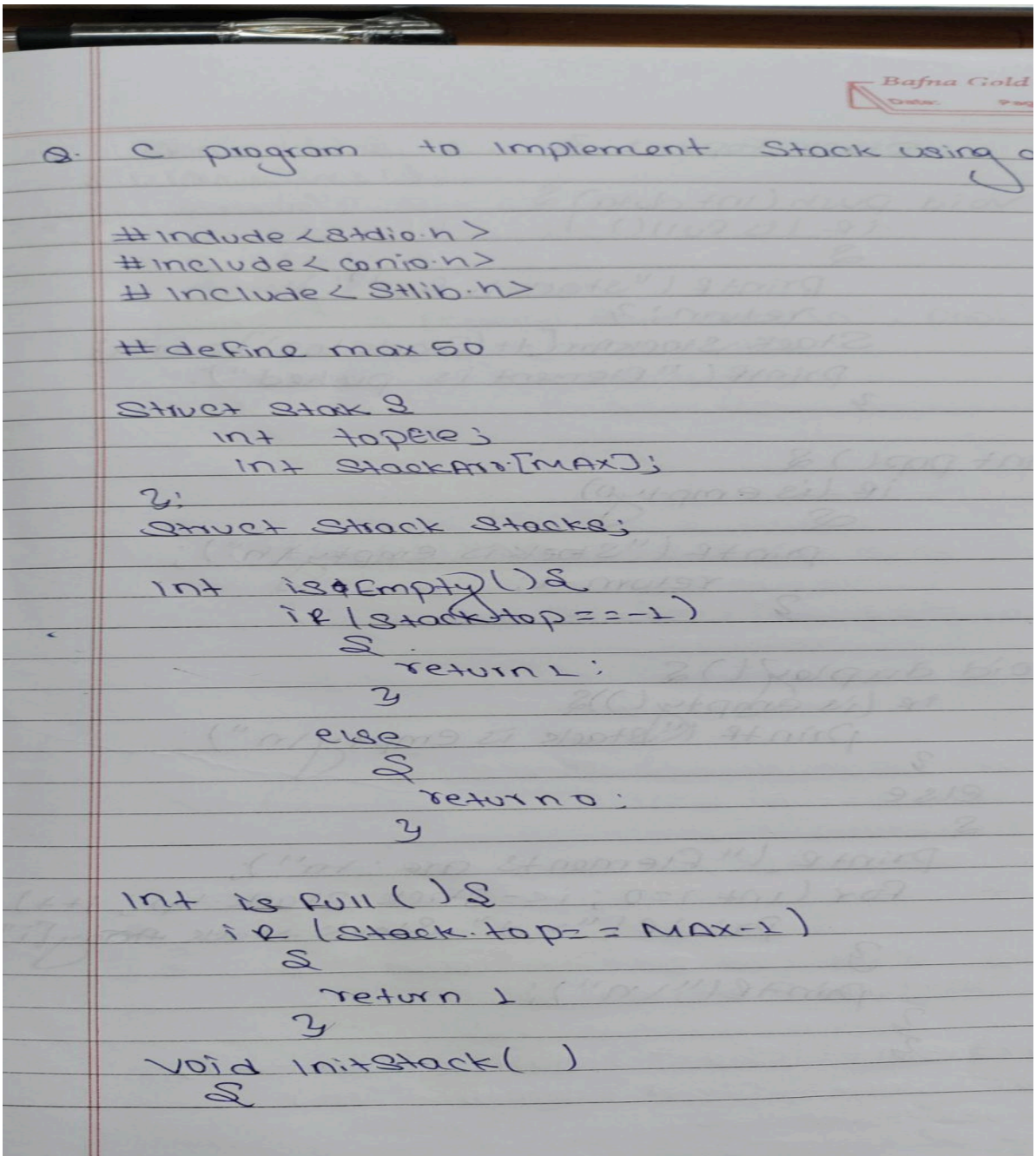
Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

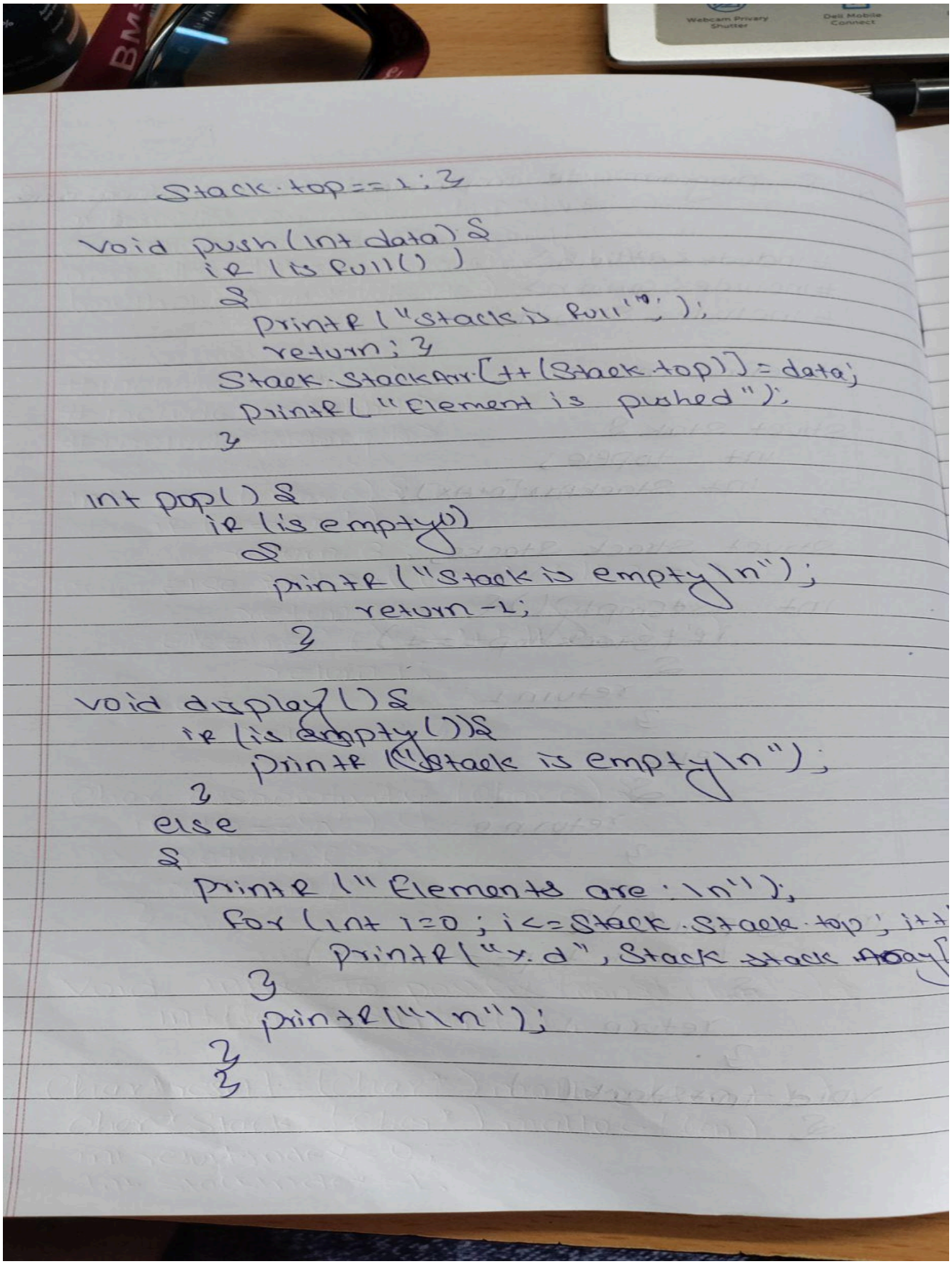
c) Display

The program should print appropriate messages for stack overflow, stack underflow

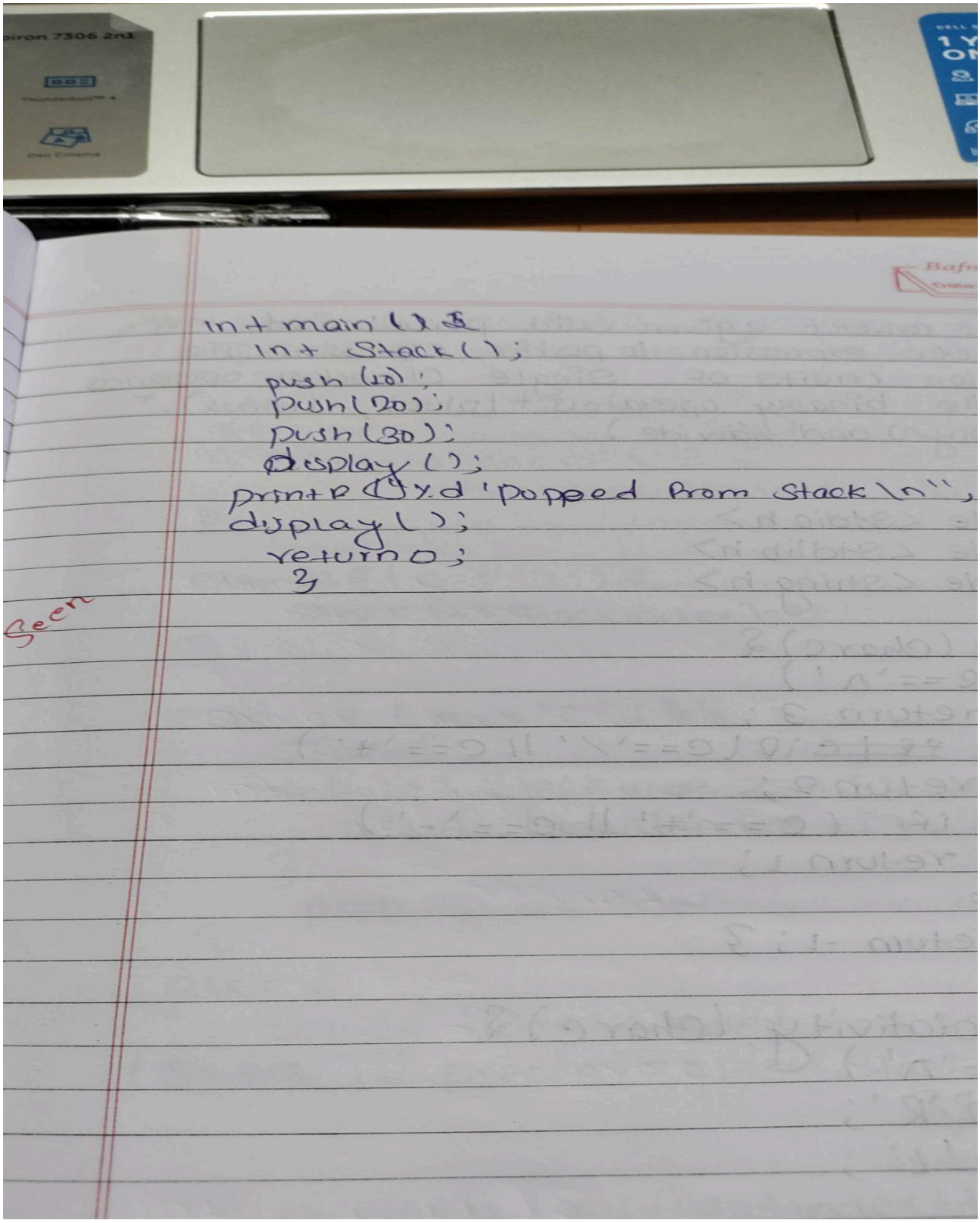


Q. C program to implement Stack using

```
#include<stdio.h>
#include<conio.h>
#include<stlib.h>

#define max 50

struct stack {
    int topele;
    int stackArr[MAX];
};
struct Stack Stacks;

int isEmpty() {
    if (Stack.top==-1)
        {
            return 1;
        }
    else
        {
            return 0;
        }

int isFull() {
    if (Stack.top==MAX-1)
        {
            return 1
        }
void initStack( )
    {
```



```
Stack.top==1;}

Void push(int data){
    if (is full() )
    {
      printf("Stack is full");
      return;}
    Stack.StackArr[++(Stack.top)] = data;
     printf("Element is pushed");
    }

int pop(){
    if (is empty())
     {
      printf("Stack is empty\n");
          return -1;
       }

void display(){
    if (is empty()){
      printf("Stack is empty\n");
   }
  else
  {
    printf("Elements are:\n");
     for(int i=0; i<=Stack.Stack.top; i++
         printf("%d", Stack.Stack Array[
     }
     printf("\n");
  }
  }
```



```
int main () {
  int stack();
  push(10);
  push(20);
  push(30);
  display();
printf("%d popped from stack\n",
display();
  return 0;
  }
```

Seen

code :

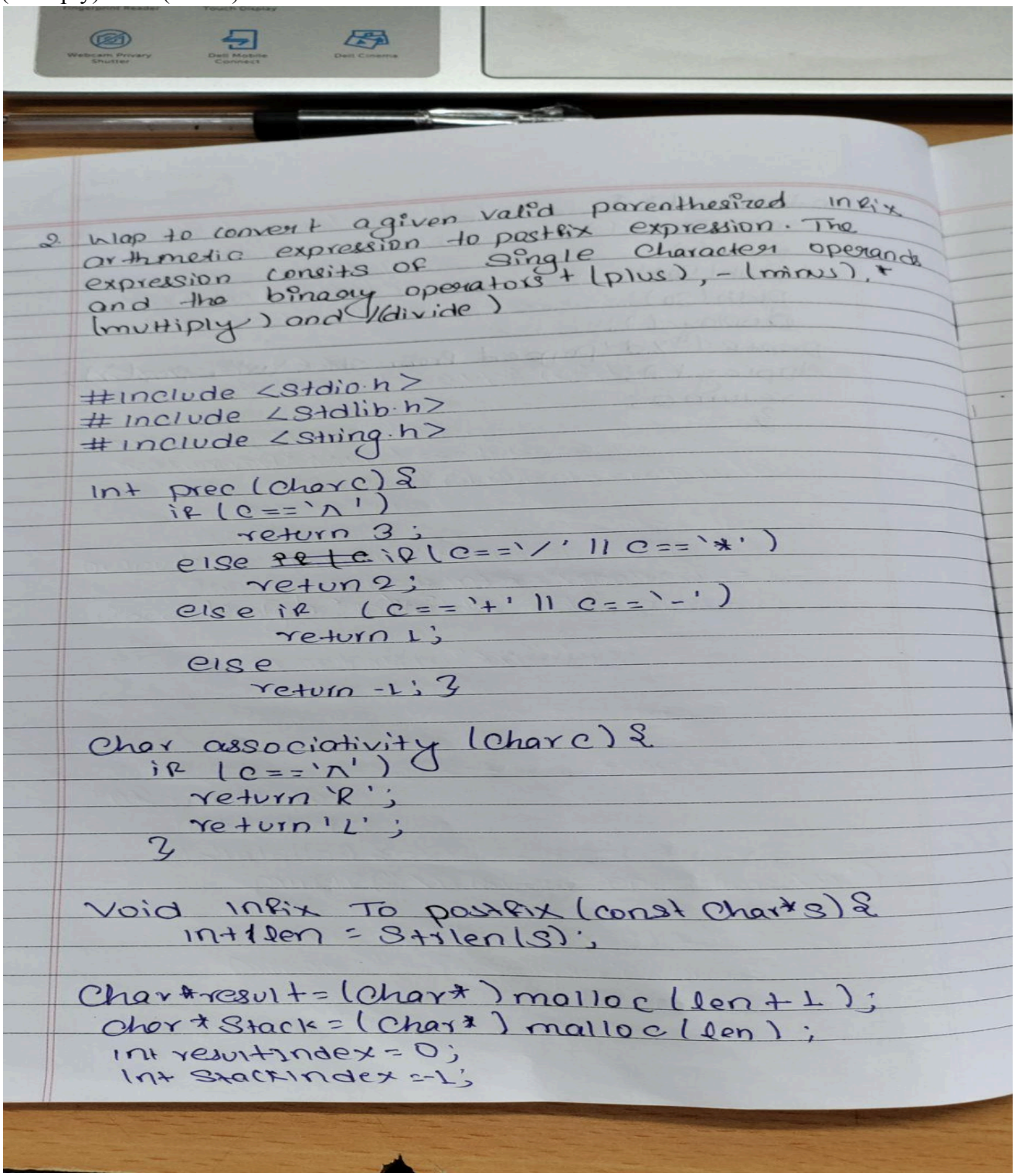
```
#include <stdio.h>
#define MAX 5
int stack[MAX];
int top = -1;

void push(int value) {
   if(top == MAX - 1) {
      printf("Stack Overflow\n");
   } else {
      top++;
      stack[top] = value;
      printf("%d pushed to stack\n", value);
   }
}

void pop() {
   if(top == -1) {
      printf("Stack Underflow\n");
   } else {
      printf("%d popped from stack\n", stack[top]);
      top--;
   }
}

void display() {
   if(top == -1) {
      printf("Stack is empty\n");
   } else {
      printf("Stack elements: ");
      for(int i = top; i >= 0; i--) {
         printf("%d ", stack[i]);
      }
      printf("\n");
   }
}

int main() {
```

```
    push(10);
    push(20);
    push(30);
    push(40);
    push(50);
    push(60);
    display();
    pop();
    pop();
    display();
    pop();
    pop();
    pop();
    pop();
    return 0;
}
```

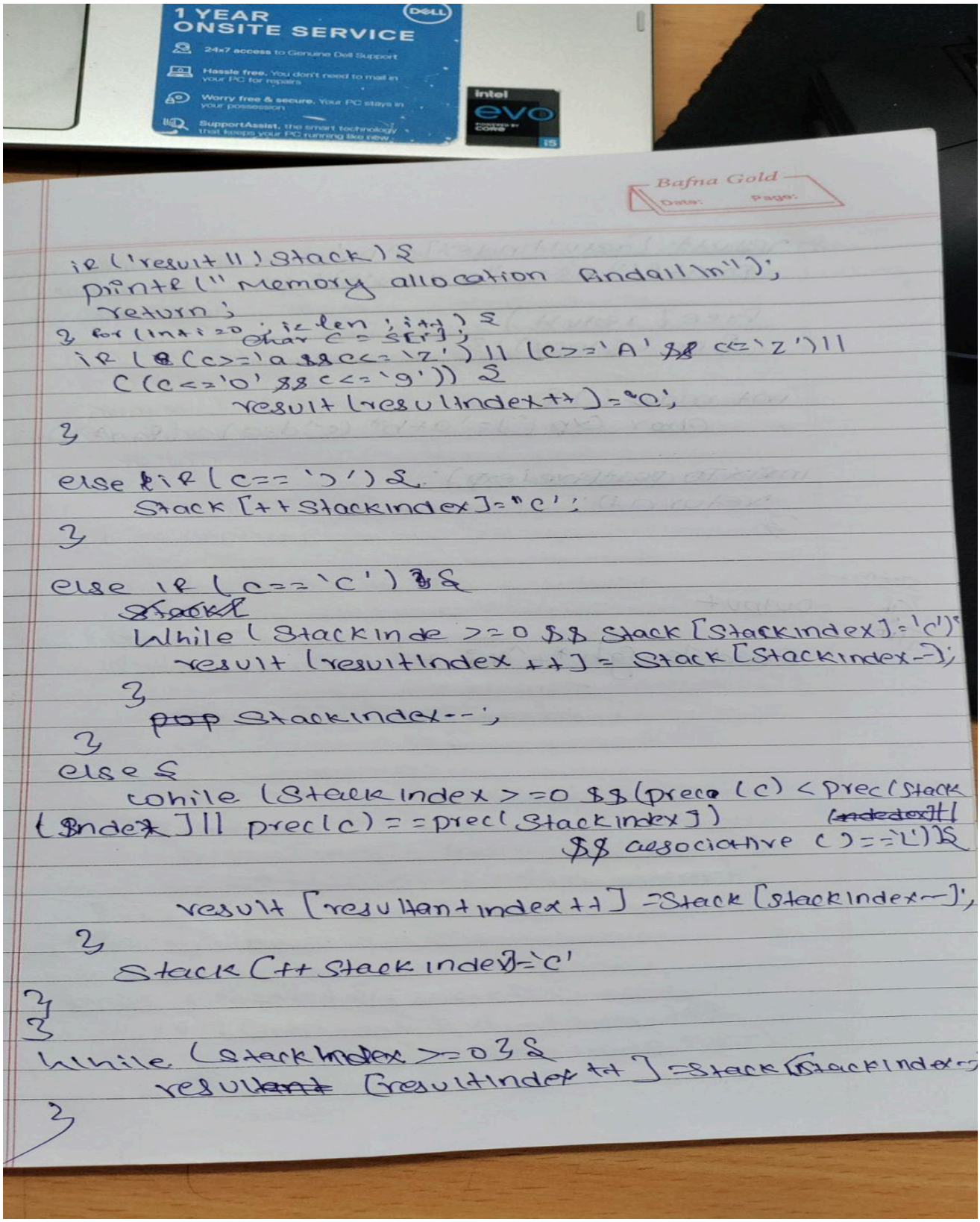
```
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\Desktop\ds\"
 -o lab1 } ; if ($?) { .\lab1 }
10 pushed to stack
20 pushed to stack
30 pushed to stack
40 pushed to stack
50 pushed to stack
Stack Overflow
Stack elements: 50 40 30 20 10
50 popped from stack
40 popped from stack
Stack elements: 30 20 10
30 popped from stack
20 popped from stack
10 popped from stack
Stack Underflow
PS C:\Users\Soham\OneDrive\Desktop\ds>
```

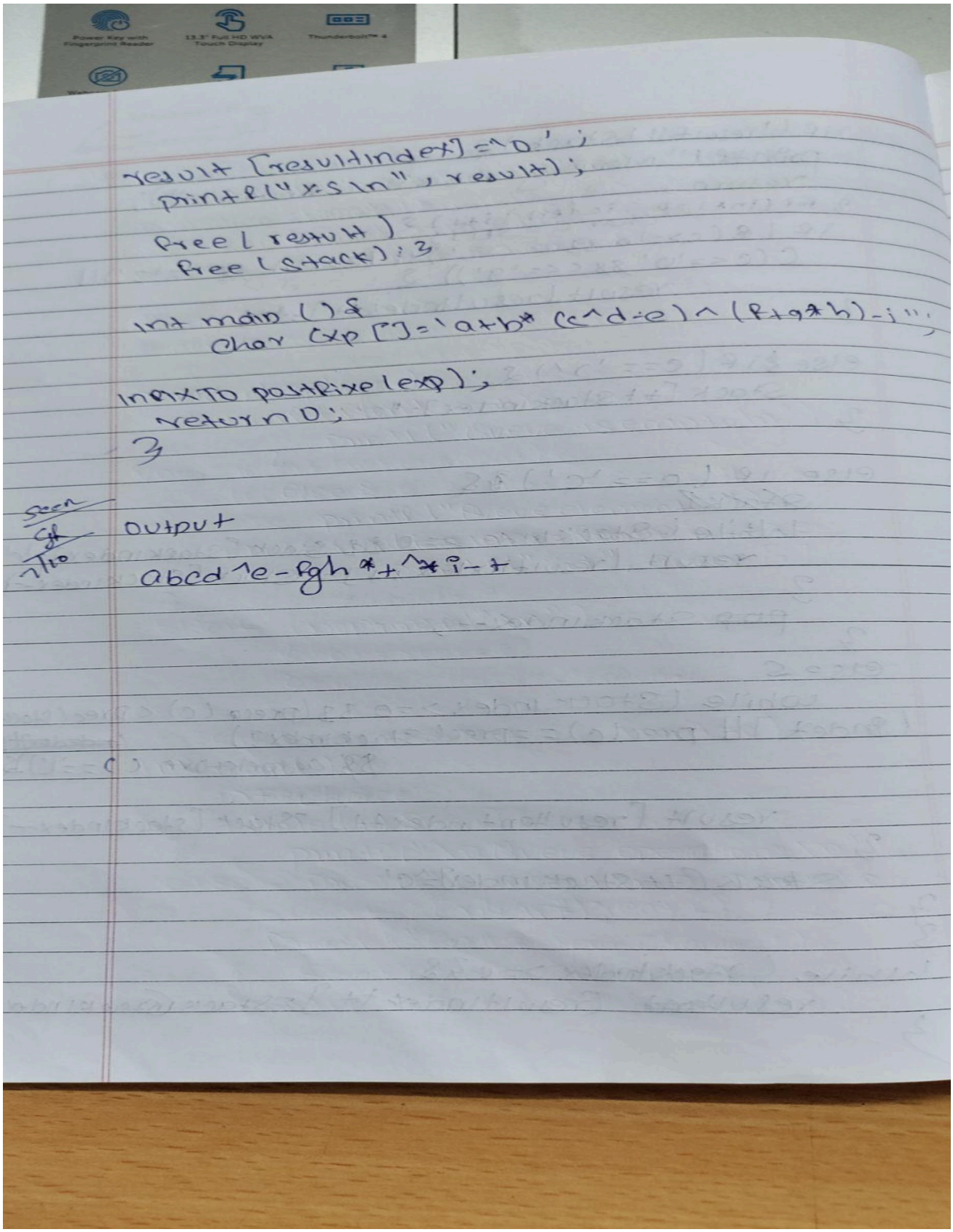
Lab programme 2

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)



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

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

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

char associativity (char c){
    if (c=='^')
     return 'R';
     return 'L';
}

Void infix To postfix (const char*s){
    int len = strlen(s);

char *result=(char*) malloc(len+1);
 char *stack=(char*) malloc(len);
  int resultIndex=0;
  int stackIndex=-1;
```


1 YEAR
ONSITE SERVICE
24x7 access to Genuine Dell Support
Hassle free. You don't need to mail in your PC for repairs
Worry free & secure. Your PC stays in your possession
SupportAssist, the smart technology that keeps your PC running like new
intel
evo
Bafna Gold
Date:
Page:
if (!result || !stack) {
printf("Memory allocation Fail\n");
return;
} for (int i=0; i<len; i++) {
char c = str[i];
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 (stackinde >=0 && stack[stackindex]!='(')
result[resultindex++] = stack[stackindex--];
}
pop stackindex--;
}
else {
while (stackindex>=0 && (prec(c)<prec(stack
[index]) || prec(c)==prec(stackindex])
&& associative()=='L')) {
result[resultant index++] =stack[stackindex--];
}
stack[++stackindex]=c
}
}
while (stackindex >=0) {
result[resultindex++]=stack[stackindex--
}



```
result [resultIndex] = '\0';
printf("%s\n", result);

free(result)
free(stack); }

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

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

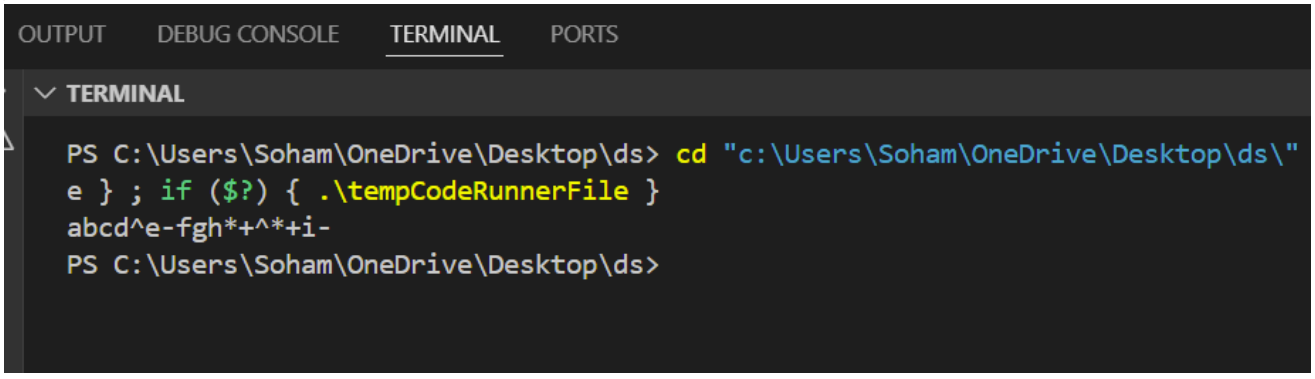
Output

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

code :

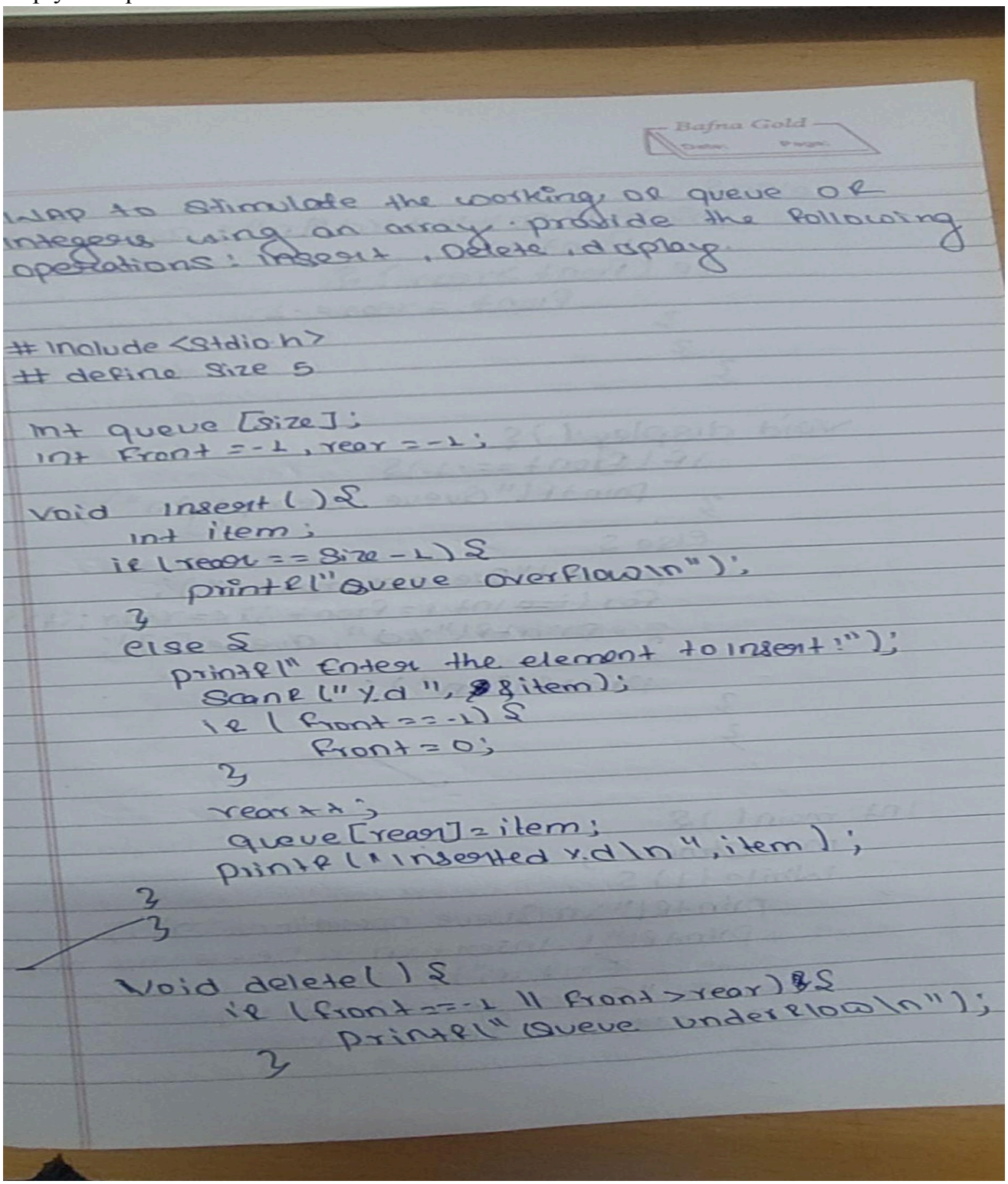
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int prec(char c) {
   if (c == '^')
      return 3;
   else if (c == '/' || c == '*')
      return 2;
   else if (c == '+' || c == '-')
      return 1;
   else
      return -1;
}
char associativity(char c) {
   if (c == '^')
      return 'R';
   return 'L';
}
void infixToPostfix(const char *expression) {
   int length = strlen(expression);
   char *result = (char *)malloc(length + 1);
   char *stack = (char *)malloc(length);
   int resultIndex = 0;
   int stackIndex = -1;

   if (!result || !stack) {
      printf("Memory allocation failed!\n");
      return;
   }

   for (int i = 0; i < length; i++) {
      char currentChar = expression[i];
       if ((currentChar >= 'a' && currentChar <= 'z') ||
         (currentChar >= 'A' && currentChar <= 'Z') ||
         (currentChar >= '0' && currentChar <= '9')) {
         result[resultIndex++] = currentChar;
      }
      else if (currentChar == '(') {
         stack[++stackIndex] = currentChar;
```

```
        }
        else if (currentChar == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
            }
            stackIndex--;
        }
        else {
            while (stackIndex >= 0 &&
                   (prec(currentChar) < prec(stack[stackIndex]) ||
                    (prec(currentChar) == prec(stack[stackIndex]) &&
associativity(currentChar) == 'L'))) {
                result[resultIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = currentChar;
        }
    }

    while (stackIndex >= 0) {
        result[resultIndex++] = stack[stackIndex--];
    }
    result[resultIndex] = '\0';
    printf("%s\n", result);
    free(result);
    free(stack);
}

int main() {
    char expression[] = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(expression);
    return 0;
}
```

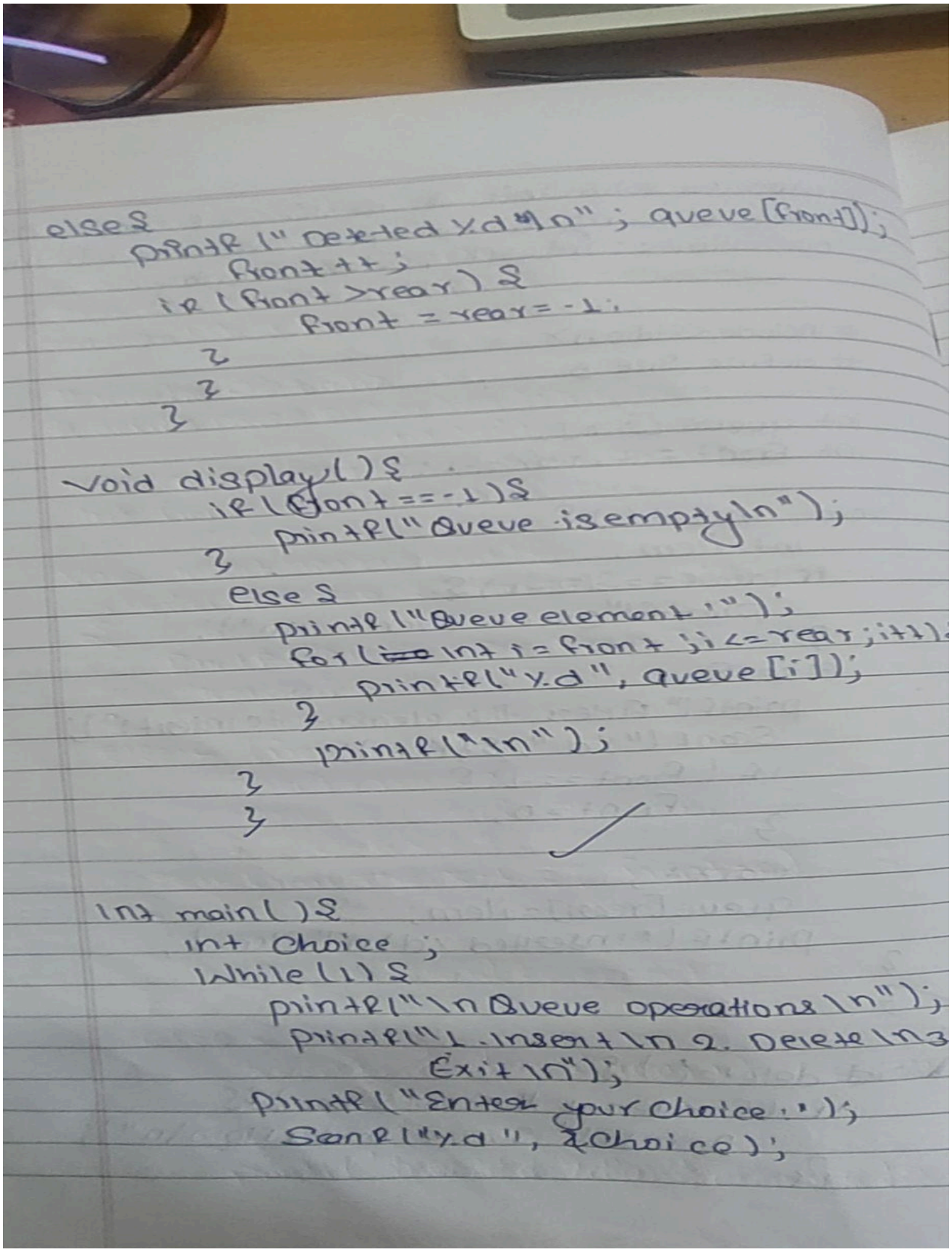


Lab programme 3

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

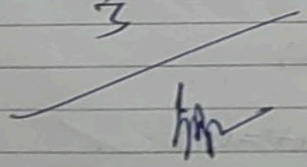


```
WAP to Stimulate the working of queue of
integers using an array. provide the following
operations: Insert, Delete, display

#include <stdio.h>
#define size 5

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

void insert(){
    int item;
    if(rear == size-1){
        printf("Queue overflow\n");
    }
    else{
        printf("Enter the element to insert:");
        scanf("%d", &item);
        if(front == -1){
            front = 0;
        }
        rear++;
        queue[rear] = item;
        printf("Inserted %d\n", item);
    }
}

Void delete(){
    if(front == -1 || front > rear){
        printf("Queue underflow\n");
    }
```



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

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

int main() {
    int choice ;
    while(1) {
        printf("\n Queue operations \n");
        printf("1. Insert \n 2. Delete \n 3
                Exit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
```

```
switch (choice){
  case 1:
      insert();
       break;

  case 2:
      delete();
       break;

  case 3:
      display();
       break;

  case 4:
      return 0;
      default:
         printf("invalid choice\n");
    }
  }
```



output:

```
queue operations
1. insert
2. delete
3. display
4. exit
enter your choice 1
enter the element to be inserted : 3
insert 3
```

code :

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

#define SIZE 5

typedef struct Queue{
    int items[SIZE];
    int front;
    int rear;
} Queue;

Queue* createQueue(){
    Queue *q = (Queue*)malloc(sizeof(Queue));

    q -> front = -1;
    q -> rear = -1;

    return q;
}



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

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


void insert(Queue *q, int value){
    if(isFull(q)){
        printf("Queue Overflow! \n");
    }

    else{
```

```
        if(isEmpty(q)){
            q -> front = 0;
        }

        q -> rear = (q -> rear + 1);
        q -> items[q -> rear] = value;

        printf("Inserted %d into Queue. \n", value);
    }
}

void delete(Queue *q){
    if(isEmpty(q)){
        printf("Queue Underflow! \n");
    }
    else{
        int removed_value = q -> items[q -> front];

        if(q -> front == q -> rear){
            q -> front = -1;
            q -> rear = -1;
        }
        else{
            q -> front = (q -> front + 1);
        }

        printf("Deleted %d from Queue. \n", removed_value);
    }
}

void display(Queue *q){
    if(isEmpty(q)){
        printf("Queue is Empty. \n");
    }
    else{
        int i = q -> front;

        printf("Queue elements: ");
```

```
        while(1){
            printf("%d, ", q -> items[i]);

            if(i == q -> rear){
                break;
            }

            i = (i + 1);
        }

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

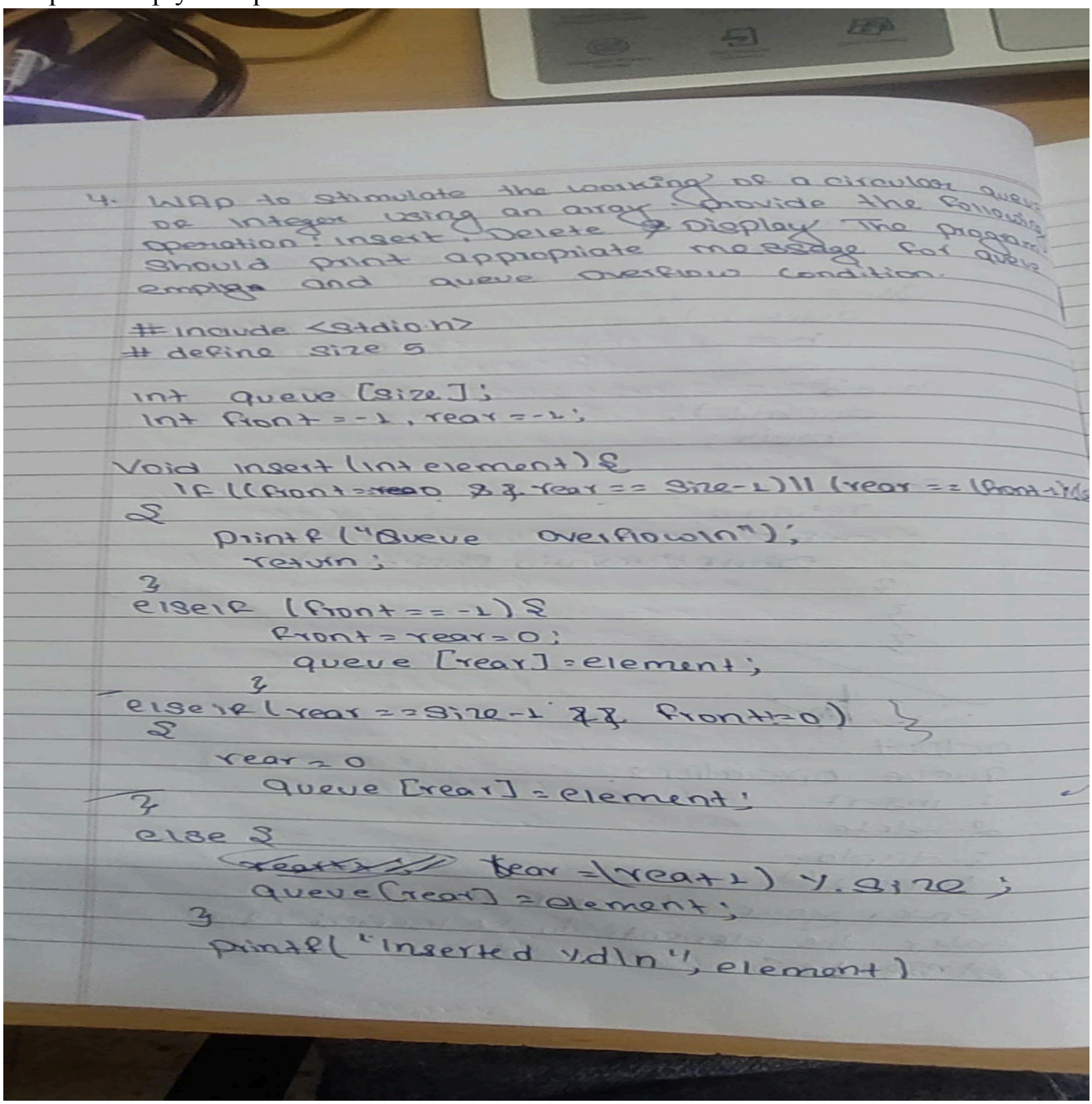
```
int main(){
    Queue *q = createQueue();
    int choice, value;

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

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

            case 2:
```

```
            delete(q);
            break;

        case 3:
            display(q);
            break;

        case 4:
            free(q);
            return 0;

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

   return 0;
}
```

```
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soha
e } ; if ($?) { .\tempCodeRunnerFile }
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 1
Enter value to insert: 3
Inserted 3 into Queue.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 1
Enter value to insert: 4
Inserted 4 into Queue.
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 3
Queue elements: 3, 4,
 1. Insert
 2. Delete
 3. Display
 4. Exit
Enter your choice: 2
Deleted 3 from Queue.
```

program 4

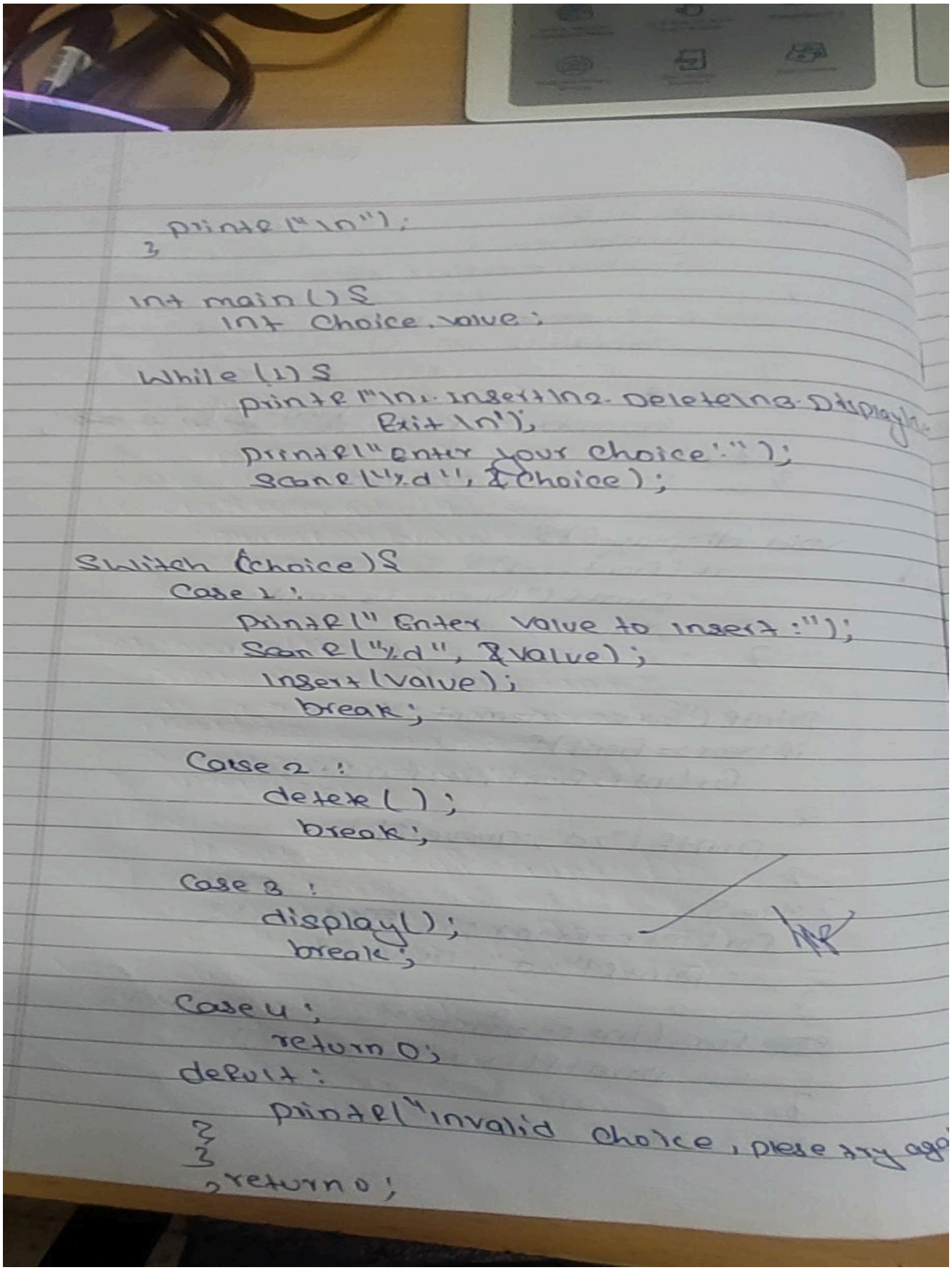
WAP to simulate the working of a circular queue of integers using an array. Provide the following opera ons: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions.



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

#include <stdio.h>
# define size 5

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

void insert (int element) {
  if ((front == rear && rear == size-1) || (rear == (front-1)%(
{
    printf ("Queue overflow\n");
     return;
}
else if (front == -1) {
     front = rear = 0;
      queue [rear] = element;
    }
else if (rear == size-1 && front != 0) {
{
    rear = 0
     queue [rear] = element;
}
else {
     rear = (rear+1) % size;
    queue[rear] = element;
}
printf("Inserted %d\n", element)
```

front %. size.

```
void delete() {
    if(front==rear) {
        front=rear=-1;
    }
    else if (front==size-1) {
        front=0;
    }
    else {
        front++, front= front % size;
    }
}

void display() {
    if (front==-1) {
        printf("Queue is empty\n");
        return;
    }

printf("Queue element:\n");
if(rear>=front) {
    for(int i=front; i<=rear; i++)
{
    printf("%d", queue[i]);
}
else {
    for(int i=front; i<=size-1; 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("\n1. Inserting 2. Deleting 3. Displaying
                4. Exit \n");
        printf("Enter 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, please try agg
    }
}
return 0;
```

code:

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

#define MAX 5

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

CircularQueue* createQueue() {
   CircularQueue* queue = (CircularQueue*)malloc(sizeof(CircularQueue));
   queue->front = -1;
   queue->rear = -1;
   return queue;
}

int isFull(CircularQueue* queue) {
   return (queue->front == (queue->rear + 1) % MAX);
}

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

void enqueue(CircularQueue* queue, int value) {
   if (isFull(queue)) {
      printf("Queue is full!\n");
      return;
   }
   if (queue->front == -1) {
      queue->front = 0;
   }
   queue->rear = (queue->rear + 1) % MAX;
   queue->items[queue->rear] = value;
   printf("Enqueued: %d\n", value);
}

int dequeue(CircularQueue* queue) {
```

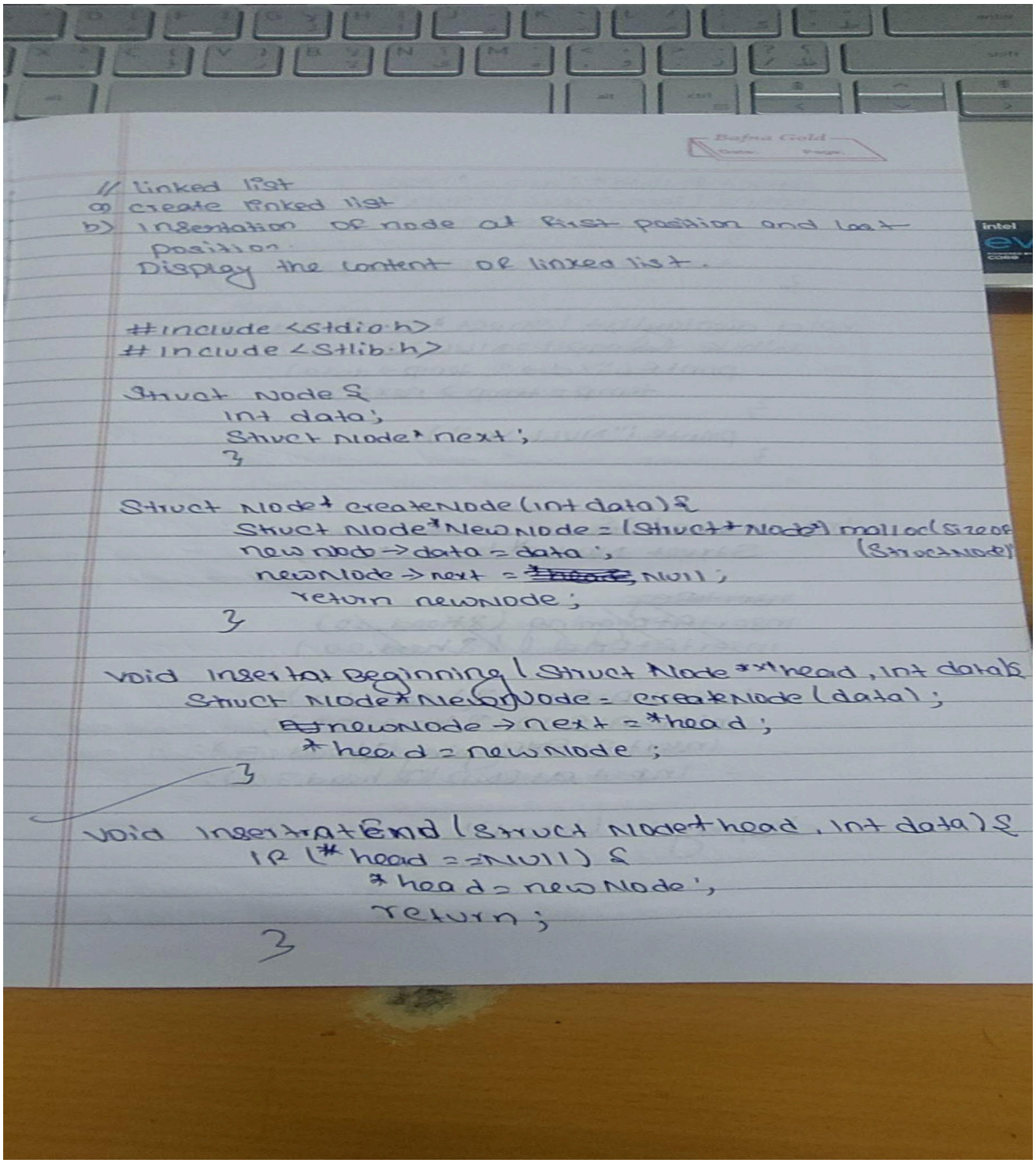
```
    if (isEmpty(queue)) {
        printf("Queue is empty!\n");
        return -1;
    }
    int value = queue->items[queue->front];
    if (queue->front == queue->rear) {
        queue->front = -1;
        queue->rear = -1;
    } else {
        queue->front = (queue->front + 1) % MAX;
    }
    printf("Dequeued: %d\n", value);
    return value;
}

void display(CircularQueue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue: ");
    for (int i = queue->front; i != queue->rear; i = (i + 1) % MAX) {
        printf("%d ", queue->items[i]);
    }
    printf("%d\n", queue->items[queue->rear]);
}

int main() {
    CircularQueue* queue = createQueue();
    int choice, value;
    do {
        printf("\nCircular Queue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(queue, value);
                break;
            case 2:
                dequeue(queue);
                break;
            case 3:
                display(queue);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    free(queue);
    return 0;
}
```

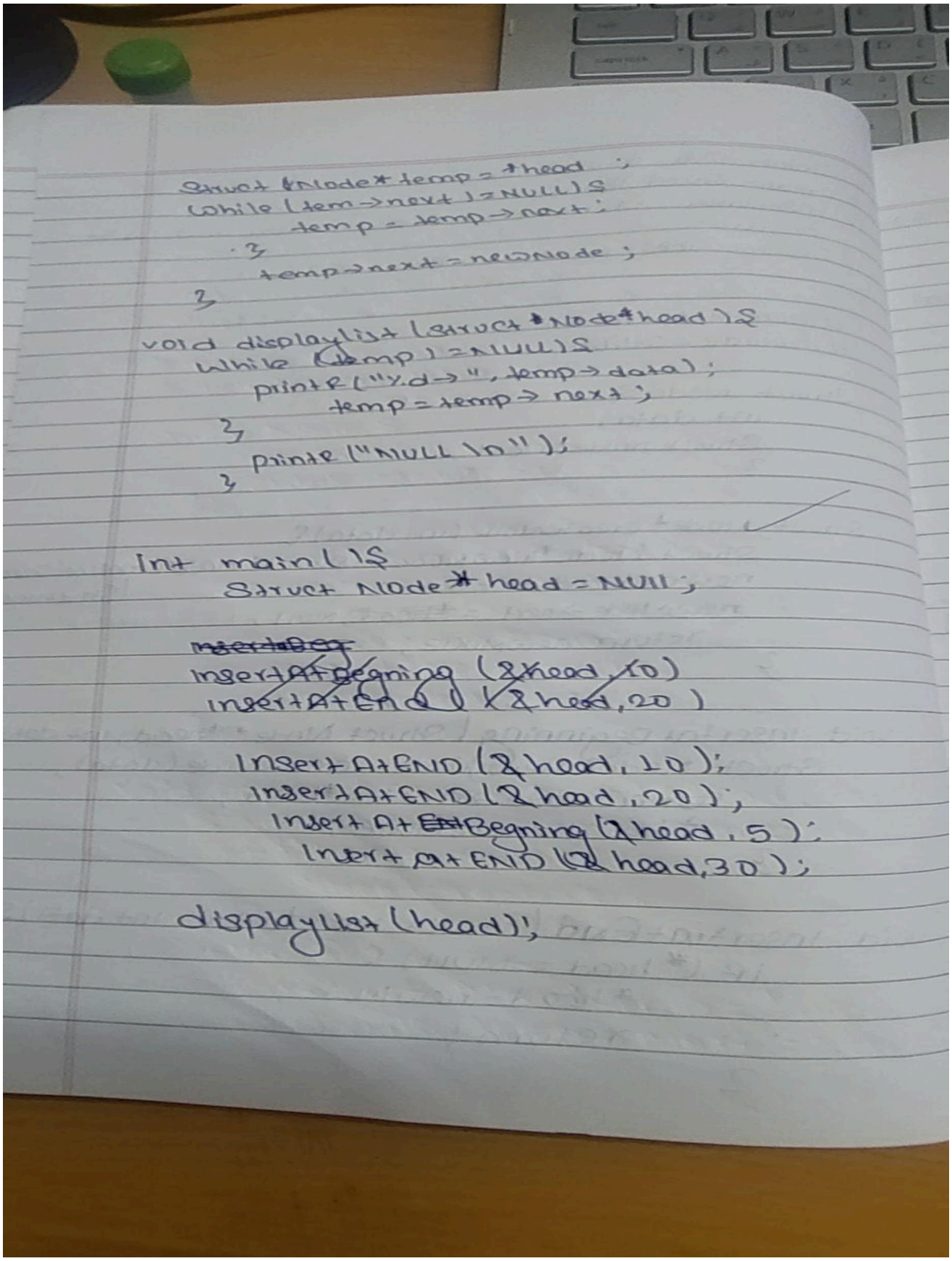
```
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\
e } ; if ($?) { .\tempCodeRunnerFile }

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 2
Enqueued: 2

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 3
Enqueued: 3

Circular Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
```

program 5

WAP to Implement Singly Linked List with following opera ons a) Createalinkedlist. b) Inser on of a node at first posi on, at any posi on and at end of list. Display the contents

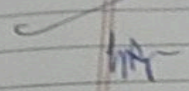


```
// linked list
a) create linked list
b) insertation of node at first position and last
   position
   Display the content of linked list.

#include <stdio.h>
#include <stlib.h>

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

Struct Node* createNode(int data){
      Struct Node* NewNode = (Struct* Node*) malloc(sizeof
                                          (Struct Node))
      new node -> data = data;
      newNode -> next = NULL;
        return newNode;
      }

void insertatBeginning(Struct Node **head, int data){
    Struct Node* NewNode = createNode(data);
        newNode -> next = *head;
        *head = newNode;
      }

void insertatEnd(Struct Node* head, int data){
      if (*head == NULL) {
            *head = newNode;
            return;
      }
```



```
Struct Node* temp = *head ;
while (tem->next != NULL) {
    temp = temp->next;
}
temp->next = newNode ;
}

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

int main() {
Struct Node * head = NULL;

insertAtBegning (&head, 10)
insertAtEnd (&head, 20)

InsertAtEND (&head, 10);
InsertAtEND (&head, 20);
Insert At Begning (&head, 5);
Insert At END (&head, 30);

displayList (head);
```

```
Struct Node* temp ;
while (head != NULL){
        temp = head;
        head = head -> next ;
        free (temp);
    }
    return 0;
}
```



code:

```
#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 insertAtBeginning(struct node **head, int data) {
   struct node *newNode = createNode(data);
   newNode->next = *head;
   *head = newNode;
   printf("Inserted at the beginning: %d\n", data);
}
void insertAtEnd(struct node **head, int data) {
   struct node *newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      printf("Inserted at the end: %d\n", data);
      return;
   }
   struct node *temp = *head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
   printf("Inserted at the end: %d\n", data);
}

void display(struct node *head) {
```

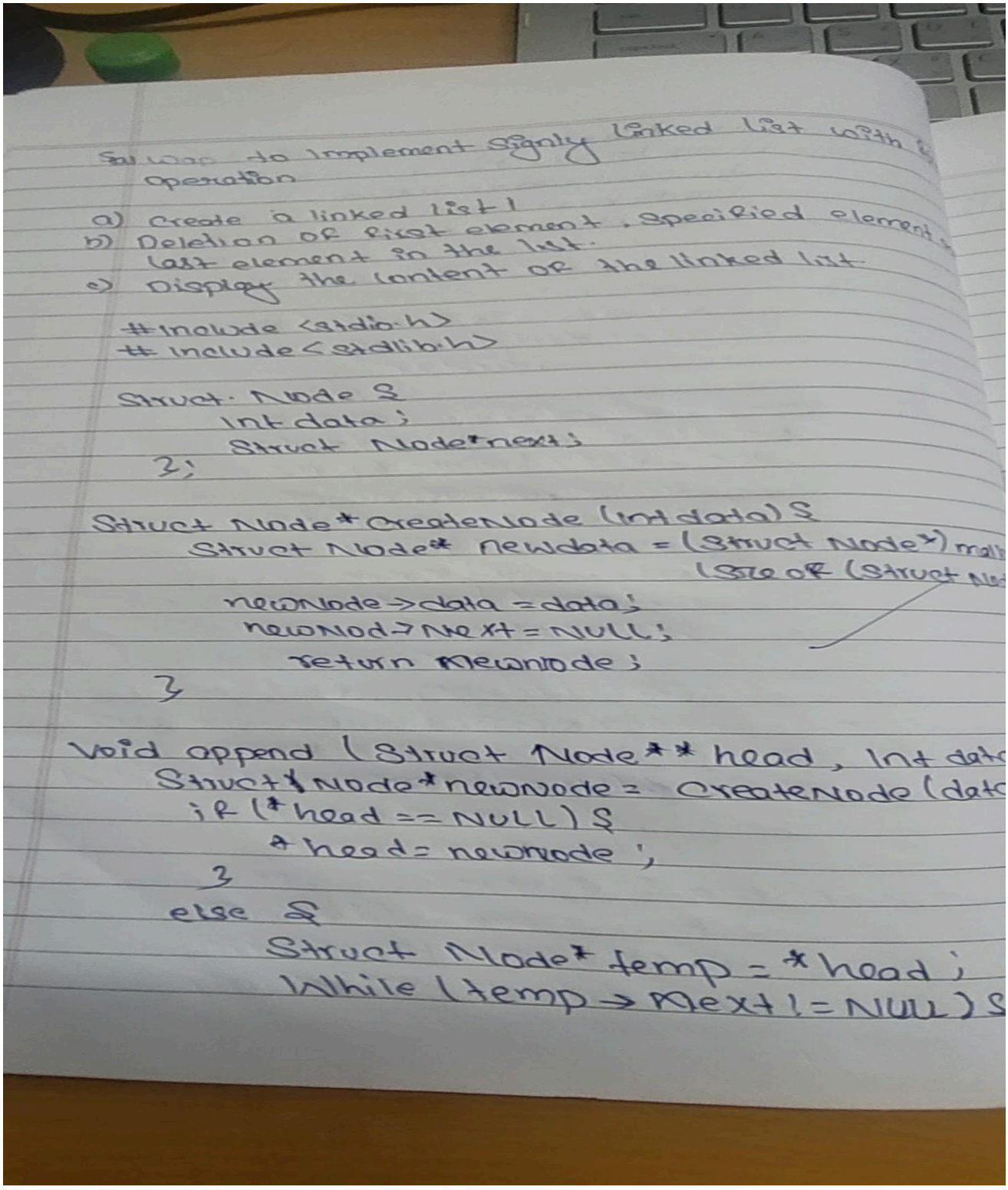
```
    struct node *temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void freeList(struct node **head) {
    struct node *temp;
    while (*head != NULL) {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}

int main() {
    struct node *head = NULL;
    int choice, data;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Display list\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
      switch (choice) {
            case 1:
                printf("Enter data to insert at beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
```

```
            insertAtEnd(&head, data);
            break;
        case 3:
            printf("Displaying the list:\n");
            display(head);
            break;
        case 4:
            freeList(&head);
            printf("Exiting the program.\n");
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
            }
 }
}
```

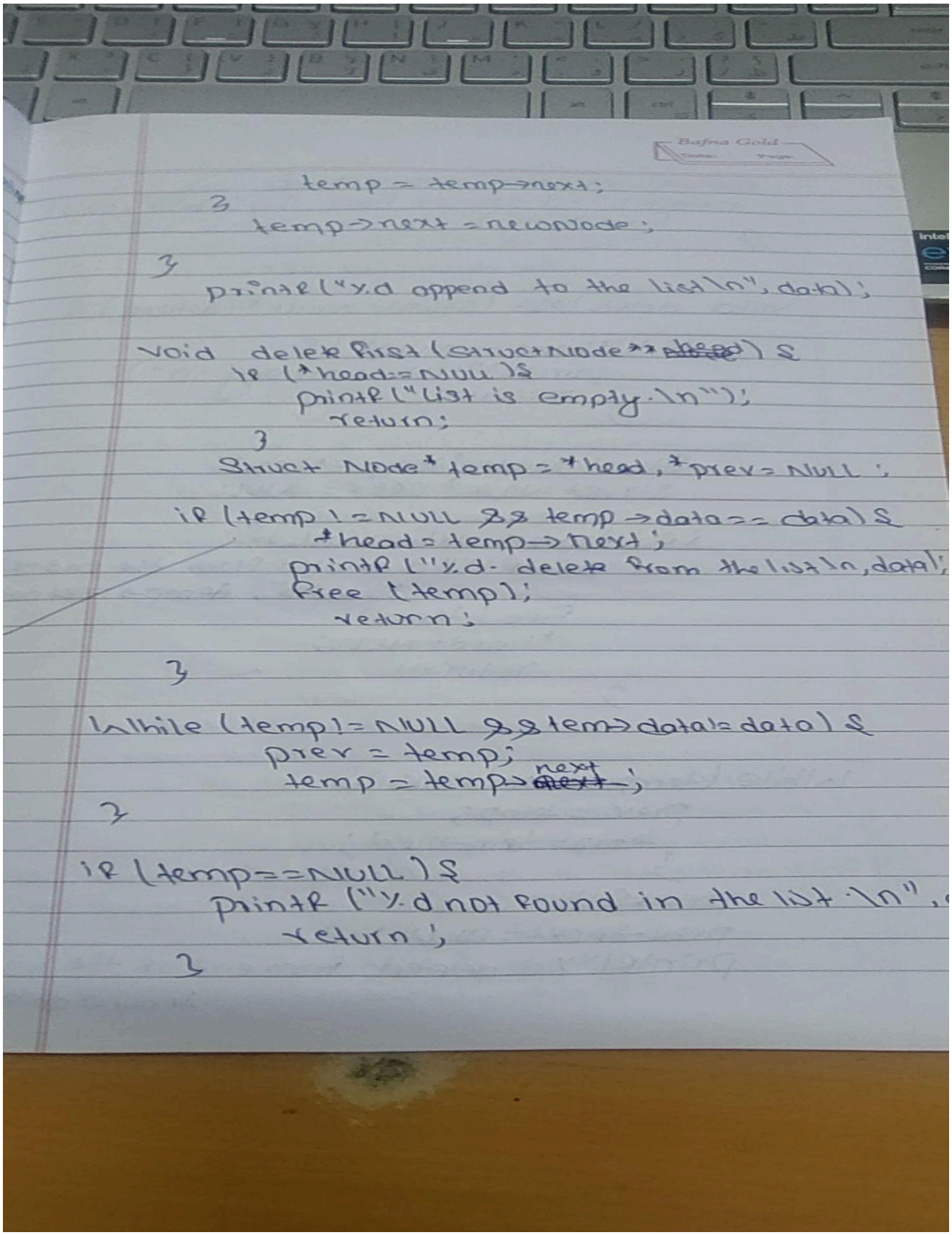
```
TERMINAL
e } ; if ($?) { .\tempCodeRunnerFile }

Menu:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice: 1
Enter data to insert at beginning: 3
Inserted at the beginning: 3

Menu:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice: 2
Enter data to insert at end: 3
Inserted at the end: 3

Menu:
1. Insert at beginning
2. Insert at end
3. Display list
4. Exit
Enter your choice: 3
Displaying the list:
3 -> 3 -> NULL
```

program 6

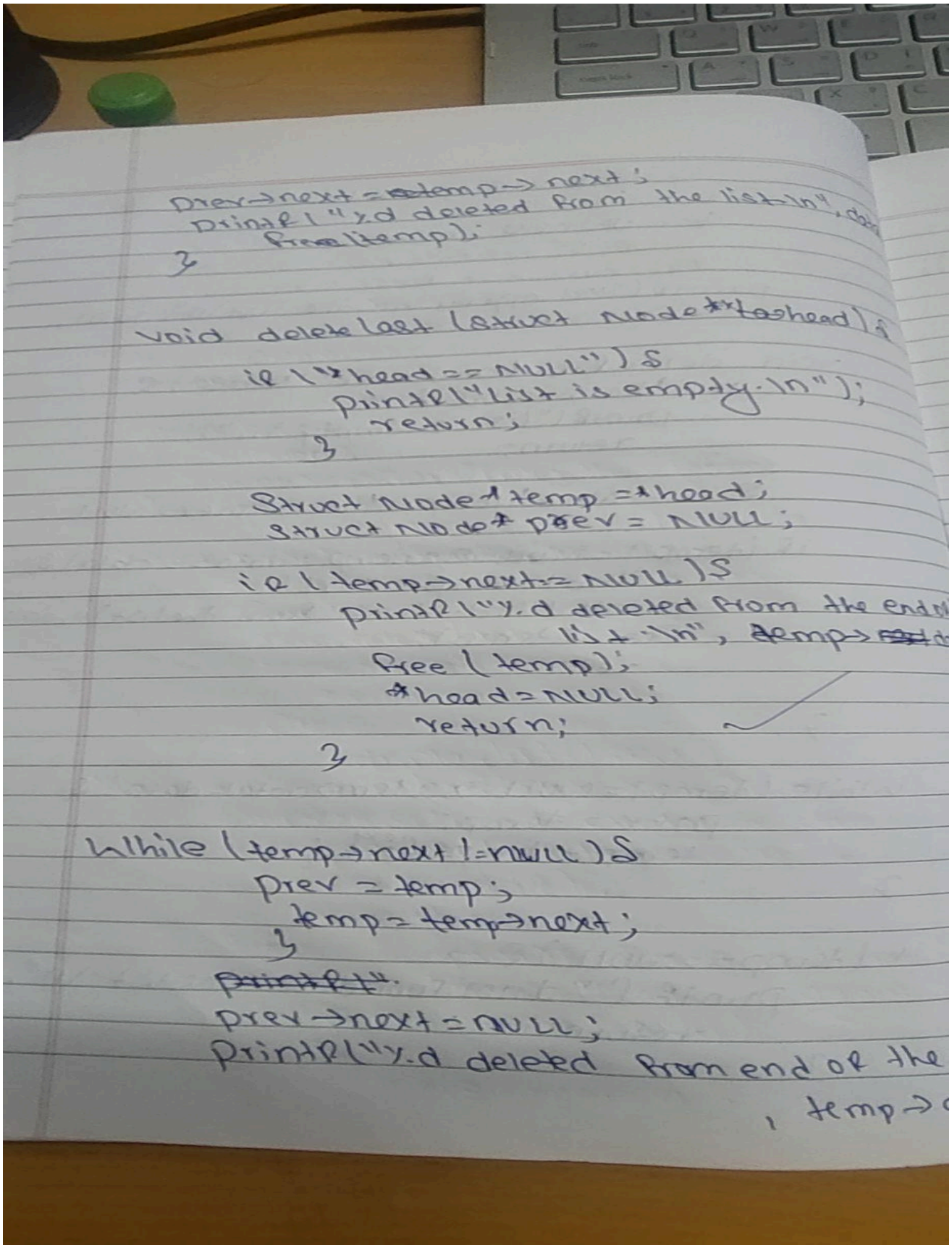
WAP to Implement Singly Linked List with following opera ons a) Create a linked list. b) Dele on of first element, specified element and last element in the list. c) Display the contents of the linked list.



```
Wap to implement Signly Linked list with
operation

a) Create a linked list
b) Deletion of first element, Specified element,
   last element in the list.
c) Display the content of the linked list.

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

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

Struct Node* CreateNode (int data) {
    Struct Node* newdata = (Struct Node*) mal
                           (Size of (Struct No

    newNode->data = data;
    newNod->Next = NULL;
    return newNode;
}

void append (Struct Node** head, int data
    Struct Node* newNode = CreateNode (data
    if (*head == NULL) {
        *head = newnode;
    }
    else {
        Struct Node* temp = *head;
        While (temp -> Next != NULL) {
```

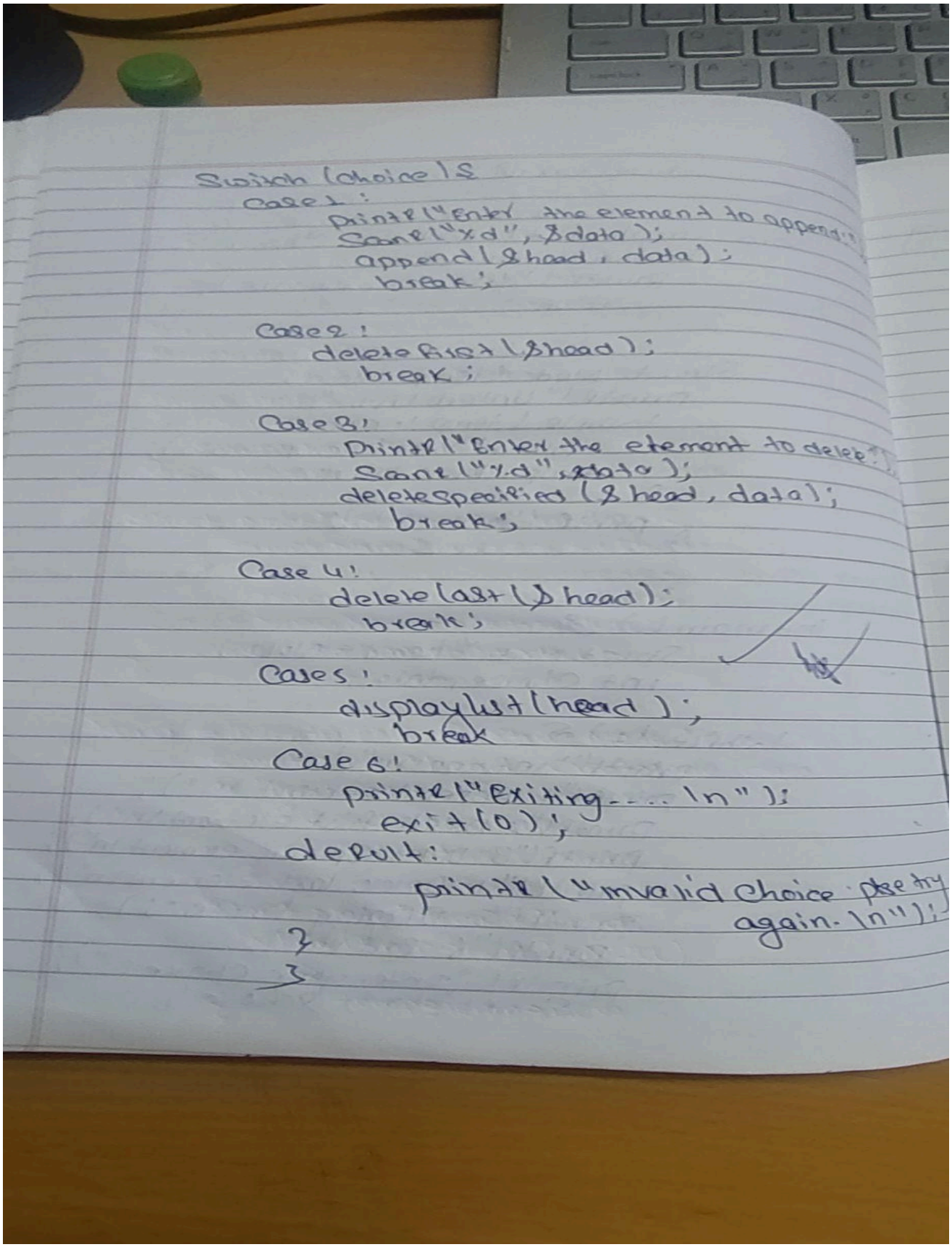


```
        temp = temp->next;
    }
    temp->next = newNode;
}
printf("%d append to the list\n", data);

void deleteFirst(struct Node **head) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node *temp = *head, *prev = NULL;

    if (temp != NULL && temp->data == data) {
        *head = temp->next;
        printf("%d. delete from the list\n", data);
        free(temp);
        return;
    }

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

if (temp == NULL) {
    printf("%d not found in the list.\n",
        return;
}
```

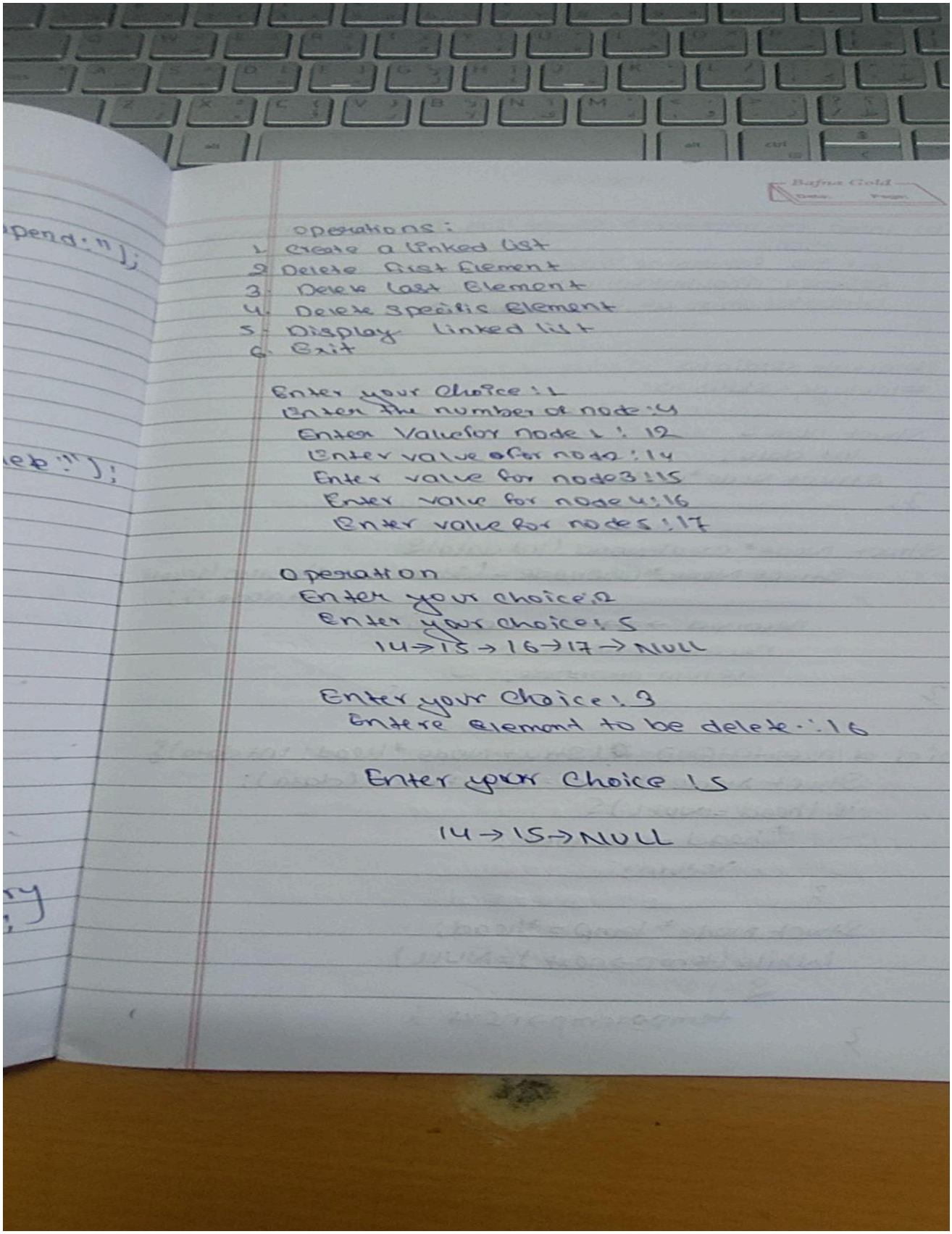


```
prev->next = temp->next;
printf("%d deleted from the list\n", data
    free(temp);
}

void deletelast(struct Node **head){
    if(*head == NULL){
        printf("List is empty.\n");
        return;
    }

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

    if(temp->next == NULL){
        printf("%d deleted from the end
                list.\n", temp->
        free(temp);
        *head = NULL;
        return;
    }

while(temp->next != NULL){
    prev = temp;
    temp = temp->next;
}
printf(".
prev->next = NULL;
printf("%d deleted from end of the
                , temp->
```

```
        free(temp);
}

void displaylist(struct Node *head){
    if(head==NULL){
        printf("List is empty.\n");
        return;
    }

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

int main(){
    struct Node *head = NULL;
    int choice, data;

while(1){
    printf("\n menu:\n");
    printf("1. Append element\n");
    printf("2. Delete first element\n");
    printf("3. Delete Specific element\n");
    printf("4. Delete last element\n");
    printf("5. Display list\n");
    printf("6. Exit\n");
    printf("Enter your choice:");
    scanf("%d", &choice);
```



```
switch (choice) {
  case 1:
    printf("Enter the element to append:");
    scanf("%d", &data);
    append(&head, data);
    break;

  case 2:
    deleteFirst(&head);
    break;

  case 3:
    printf("Enter the element to delete:");
    scanf("%d", &data);
    deleteSpecified(&head, data);
    break;

  case 4:
    deleteLast(&head);
    break;

  case 5:
    displayList(head);
    break;

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

  default:
    printf("Invalid choice. please try
                          again.\n");
  }
}
```



operations:
1. Create a linked list
2. Delete first Element
3. Delete last Element
4. Delete specific Element
5. Display linked list
6. Exit

Enter your choice: 1
Enter the number of node: 4
Enter value for node 1: 12
Enter value for node 2: 14
Enter value for node 3: 15
Enter value for node 4: 16
Enter value for node 5: 17

Operation
Enter your choice: 2
Enter your choice: 5
14 → 15 → 16 → 17 → NULL

Enter your choice: 3
Enter element to be delete: 16

Enter your choice 5

14 → 15 → NULL

code:

```
// Singly Linked List



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


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



void createLL(struct Node** head, int n){
   struct Node* newNode, *temp;
   int value;

   for(int i = 0; i < n; i++){
      newNode = (struct Node*) malloc (sizeof(struct Node));

      if (newNode == NULL){
         printf("Memory allocation failed. \n");
         return;
      }

      printf("Enter value for node %d: ", i+1);
      scanf("%d", &value);

      newNode -> data = value;
      newNode -> next = NULL;

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

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



void deleteF(struct Node** head){
    if(*head == NULL){
        printf("The list is empty. \n");
        return;
    }

    struct Node* temp = *head;
    *head = (*head) -> next;

    free(temp);
    printf("First element deleted. \n");
}



void deleteL(struct Node** head){
    if(*head == NULL){
        printf("The list is already empty. \n");
        return;
    }

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

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

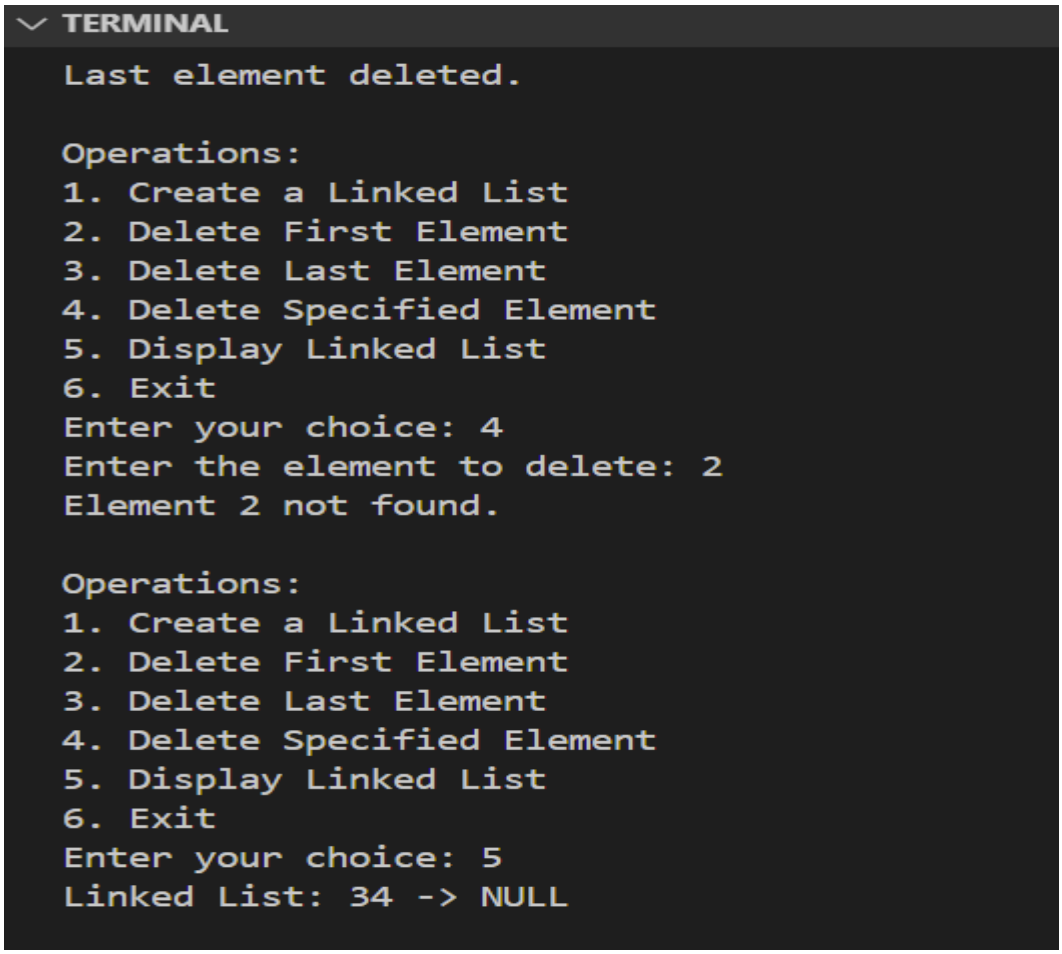
```
        printf("Last element deleted. \n");
        return;
    }

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

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




void deleteS(struct Node** head, int value){
    if(*head == NULL){
        printf("The list is already empty. \n");
        return;
    }

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

    if(temp != NULL && temp -> data == value){
        *head = temp -> next;
        free(temp);
        printf("Element %d deleted. \n", value);
        return;
    }

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

    if(temp == NULL){
```

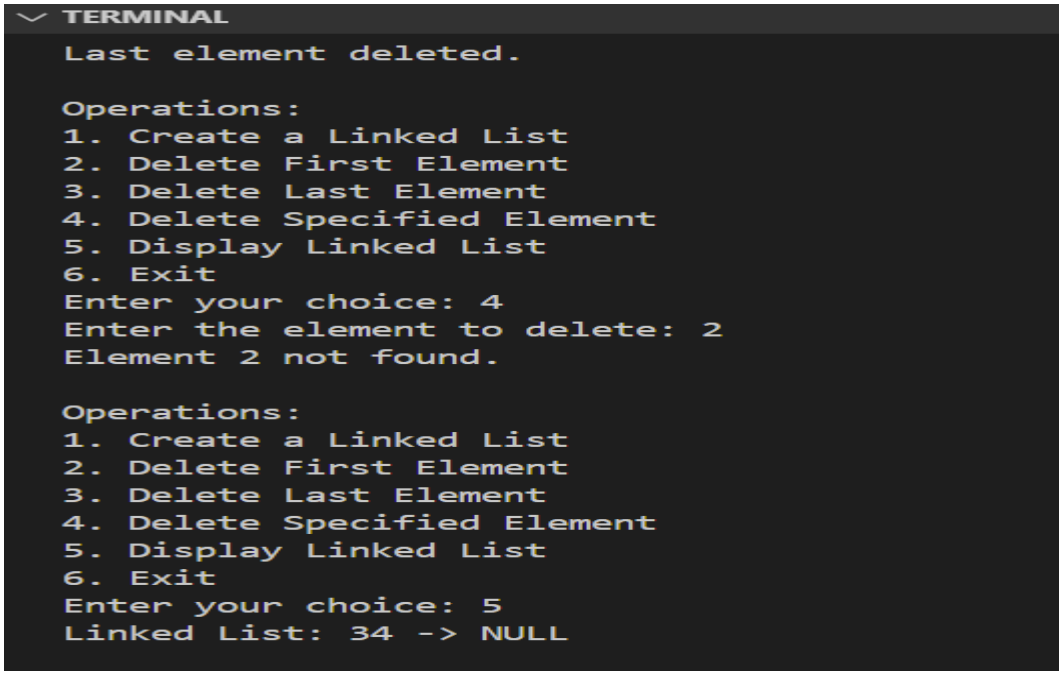
```
        printf("Element %d not found. \n", value);
        return;
    }

    prev -> next = temp -> next;
    free(temp);
    printf("Element %d deleted. \n", value);
}




void display(struct Node* head){
    if(head == NULL){
        printf("The list is already empty. \n");
        return;
    }

    struct Node* temp = head;

    printf("Linked List: ");

    while (temp != NULL)
    {
        printf("%d -> ", temp -> data);
        temp = temp -> next;
    }

    printf("NULL \n");

}




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

```
int choice, value, n;

do{
   printf("\nOperations: \n");
   printf("1. Create a Linked List \n");
   printf("2. Delete First Element \n");
   printf("3. Delete Last Element \n");
   printf("4. Delete Specified Element \n");
   printf("5. Display Linked List \n");
   printf("6. Exit \n");

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


   switch(choice){
      case 1:
         printf("Enter the number of nodes: ");
         scanf("%d", &n);
         createLL(&head, n);
         break;

      case 2:
         deleteF(&head);
         break;

      case 3:
         deleteL(&head);
         break;

      case 4:
         printf("Enter the element to delete: ");
         scanf("%d", &value);
         deleteS(&head, value);
         break;

      case 5:
         display(head);
```

```
            break;

        case 6:
            printf("Exiting program. \n");
            break;

        default:
            printf("Invalid choice!");
        }
    } while(choice != 6);

    return 0;
}
```

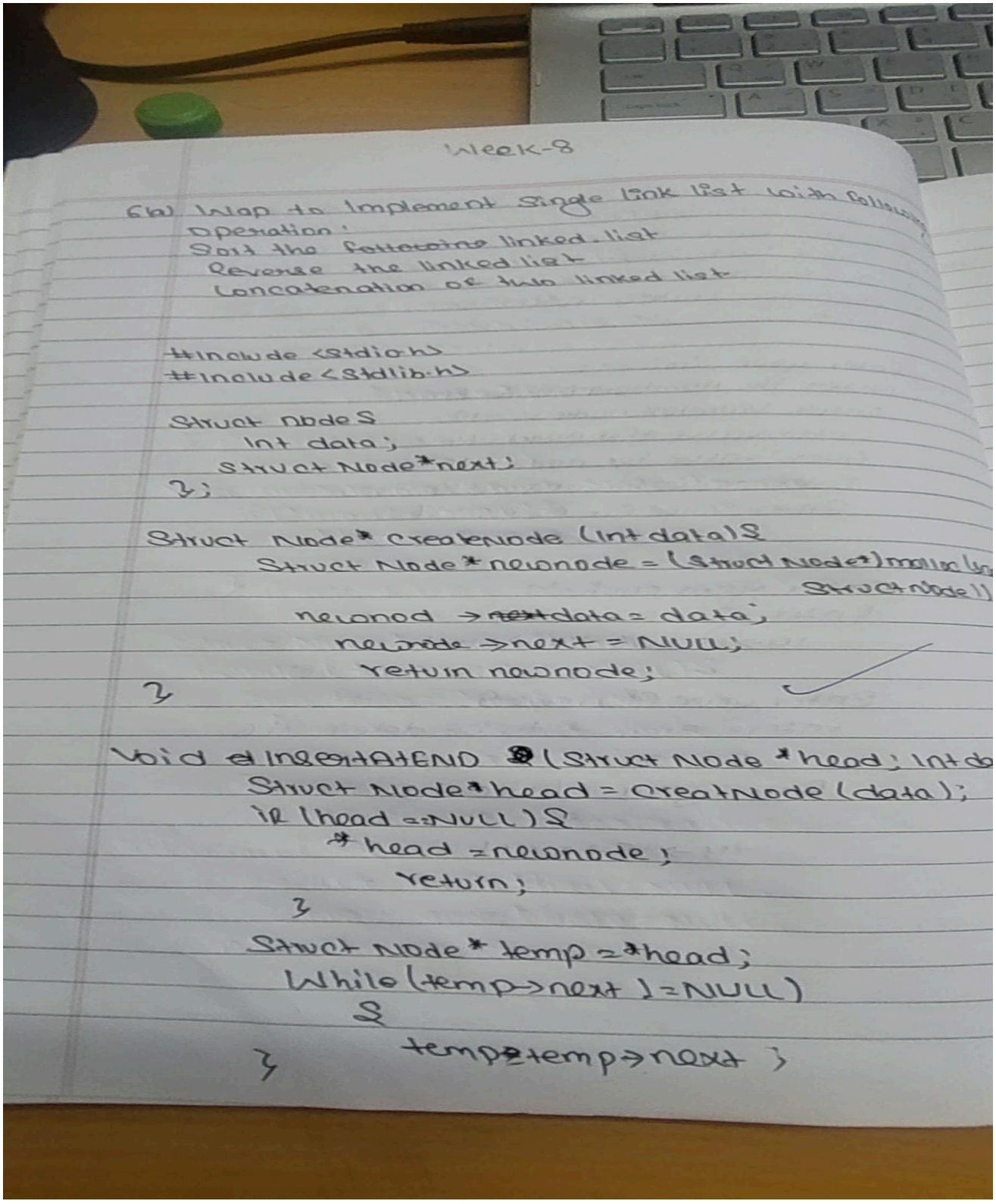
```
TERMINAL
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\De
e } ; if ($?) { .\tempCodeRunnerFile }

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the number of nodes: 3
Enter value for node 1: 2
Enter value for node 2: 34
Enter value for node 3: 4

Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 2
First element deleted.
```

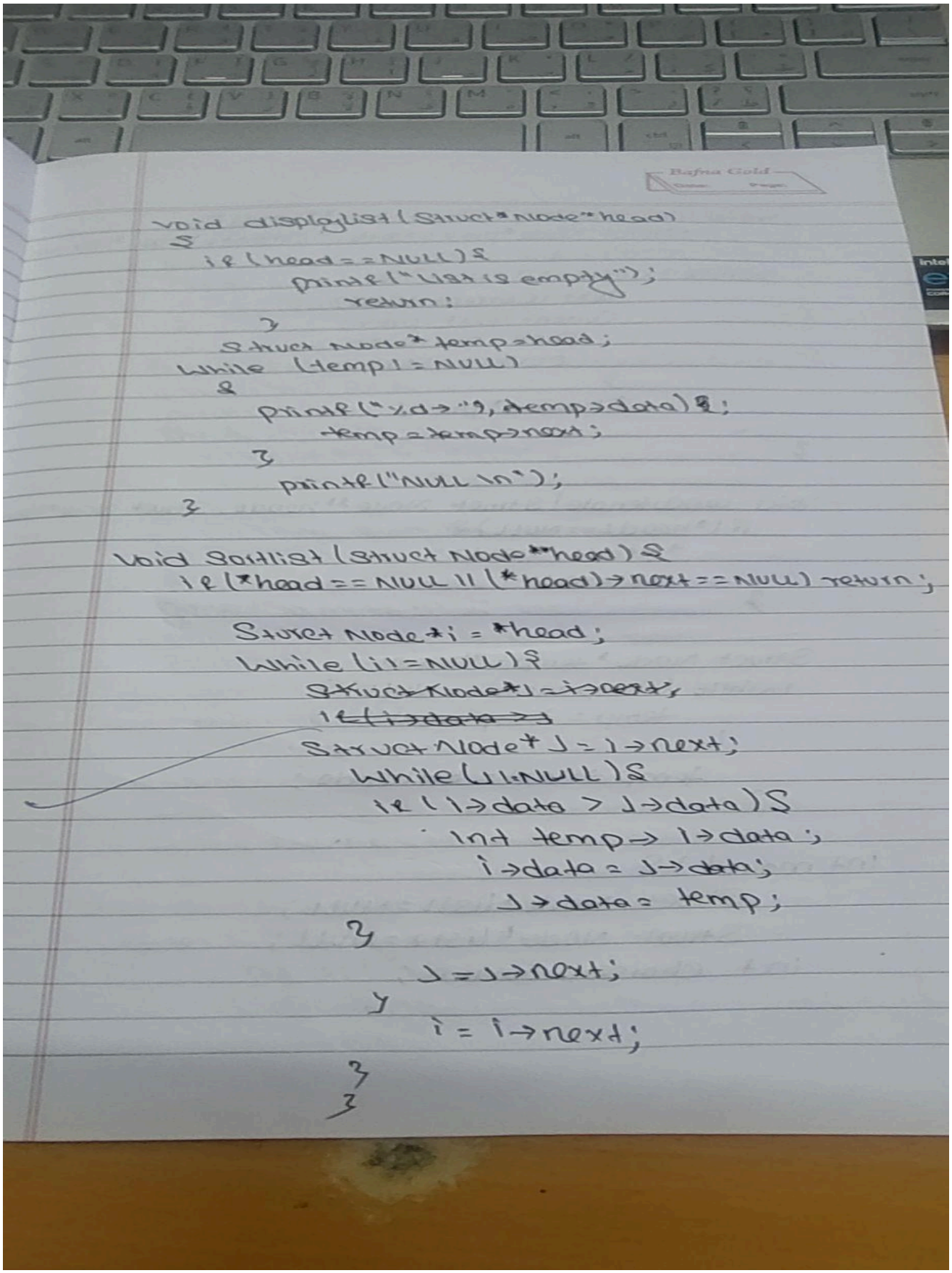

TERMINAL
Last element deleted.
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 4
Enter the element to delete: 2
Element 2 not found.
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 34 -> NULL


TERMINAL
Last element deleted.
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 4
Enter the element to delete: 2
Element 2 not found.
Operations:
1. Create a Linked List
2. Delete First Element
3. Delete Last Element
4. Delete Specified Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 34 -> NULL

program 6

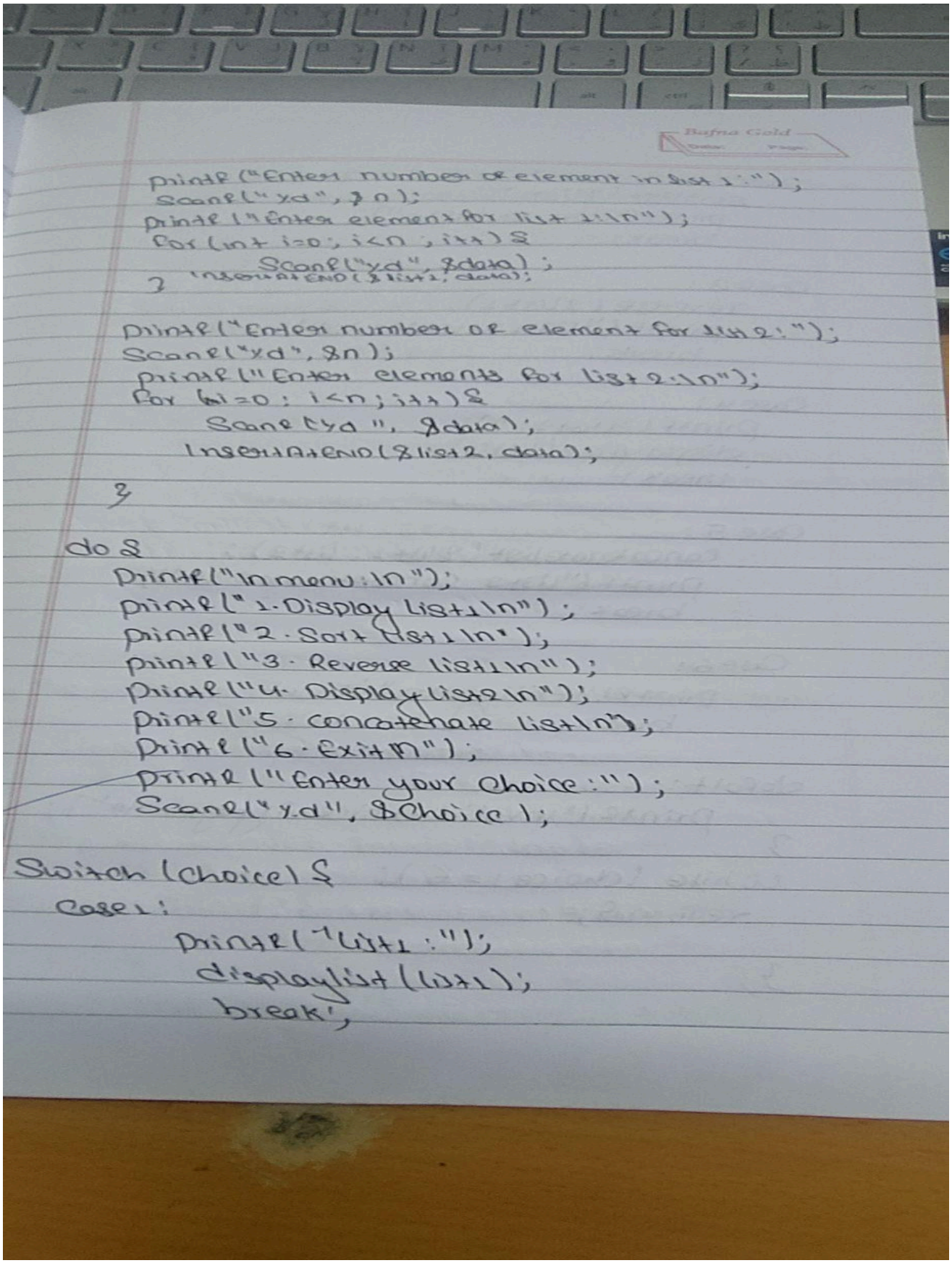
WAP to Implement Single Link List with following opera ons: Shorted linkedlist, Reversethelinkedlist, Concatena on of two linked lists





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

void sortlist(struct Node **head){
  if(*head==NULL||(*head)->next==NULL) return;

    struct Node *i=*head;
    while(i!=NULL){
      struct Node *j=i->next;
      if(i->data>j
      struct Node* j=i->next;
       while(j!=NULL){
        if(i->data > j->data){
          int 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, *current=*head;

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

}

void concatenate(Struct Node**head1, Struct
    if(*head1==NULL){
        *head1=*head2;
        return;
    }

Struct Node*temp=*head1;
while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=head2;
}

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



```
printf("Enter number of element in list 1:");
scanf("%d", &n);
printf("Enter elements for list 1:\n");
for(int i=0; i<n; i++){
    scanf("%d", &data);
    insertAtEND(&list1, data);
}

printf("Enter number of element for list 2:");
scanf("%d", &n);
printf("Enter elements for list 2:\n");
for(i=0; i<n; i++){
    scanf("%d", &data);
    insertAtEND(&list2, data);
}

do{
    printf("\n menu:\n");
    printf("1. Display list1\n");
    printf("2. Sort list1\n");
    printf("3. Reverse list1\n");
    printf("4. Display list2\n");
    printf("5. concatenate list1\n");
    printf("6. Exit\n");
    printf("Enter your choice:");
    scanf("%d", &choice);

switch(choice){
  case1:
        printf("list1:");
        displaylist(list1);
        break;
```

```
case 2:
    sortList(&list1);
    printf("List1 Sorted.\n");
    break;

case 3:
    reverse(&list1);
    printf("List1 reversed.\n");
    break;

case 4:
    printf("List2: ");
    displayList(list2);
    break;

case 5:
    concatenateList(&list1, list2);
    printf("Lists concatenated.\n");
    break;

case 6:
    printf("Exiting.\n");
    break;

default:
    printf("Invalid choice. Try again.\n");
}
while (choice != 6);
return 0;
}
```

Program -1 output:

Enter number of element in list 1 : 3
Enter the elements of list1:
1 2 3
Enter number of element in list2 : 4
Enter the elements in list2:
1 2 3 4

Menu:
1. Display list1
2. Sort list1
3. Reverse List1
4. Display list2
5. Concanate list1
6. Exit
Enter your choice : 2
List1 Sorted.
Menu:
Enter your choice: 1
List1: 1→2→3→NULL

Menu: 3
Enter your choice: 3
List1 reversed.

Menu:
Enter your choice: 1
List1 : 3→2→1→NULL

Menu:
Enter your choice : 4
List1 Concatenated.

code:

```
#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 append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void printList(struct Node* head) {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("The list is empty.\n");
        return;
    }
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
void sortList(struct Node* head) {
    struct Node* current = head;
    struct Node* index = NULL;
    int temp;

    if (head == NULL) {
        return;
    }
    while (current != NULL) {
        index = current->next;
        while (index != NULL) {
            if (current->data > index->data) {

                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->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;
```

```
    } else {
        struct Node* temp = *head1;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = head2;
    }
}
int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int choice, data, n;

    do {
        printf("\nMenu:\n");
        printf("1. Append elements to the first linked list\n");
        printf("2. Append elements to the second linked list\n");
        printf("3. Sort the first linked list\n");
        printf("4. Reverse the first linked list\n");
        printf("5. Concatenate the two linked lists\n");
        printf("6. Print the first linked list\n");
        printf("7. Print the second linked list\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of elements for the first linked list: ");
                scanf("%d", &n);
                printf("Enter the elements for the first linked list:\n");
                for (int i = 0; i < n; i++) {
                    scanf("%d", &data);
                    append(&list1, data);
                }
                break;

            case 2:
```

```
    printf("Enter the number of elements for the second linked list: ");
    scanf("%d", &n);
    printf("Enter the elements for the second linked list:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        append(&list2, data);
    }
    break;

case 3:
    sortList(list1);
    printf("The first linked list has been sorted.\n");
    break;

case 4:
    reverseList(&list1);
    printf("The first linked list has been reversed.\n");
    break;

case 5:
    concatenateLists(&list1, list2);
    printf("The two linked lists have been concatenated.\n");
    break;

case 6:
    printf("First Linked List: ");
    printList(list1);
    break;

case 7:
    printf("Second Linked List: ");
    printList(list2);
    break;

case 8:
    printf("Exiting the program.\n");
    break;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 8);

    return 0;
}
```

```
TERMINAL
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\Desktop\ds\"
e } ; if ($?) { .\tempCodeRunnerFile }

Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
6. Print the first linked list
7. Print the second linked list
8. Exit
Enter your choice: 1
Enter the number of elements for the first linked list: 3
Enter the elements for the first linked list:
1 2 4

Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
6. Print the first linked list
7. Print the second linked list
8. Exit
Enter your choice: 2
Enter the number of elements for the second linked list: 3
Enter the elements for the second linked list:
5 6 7
```

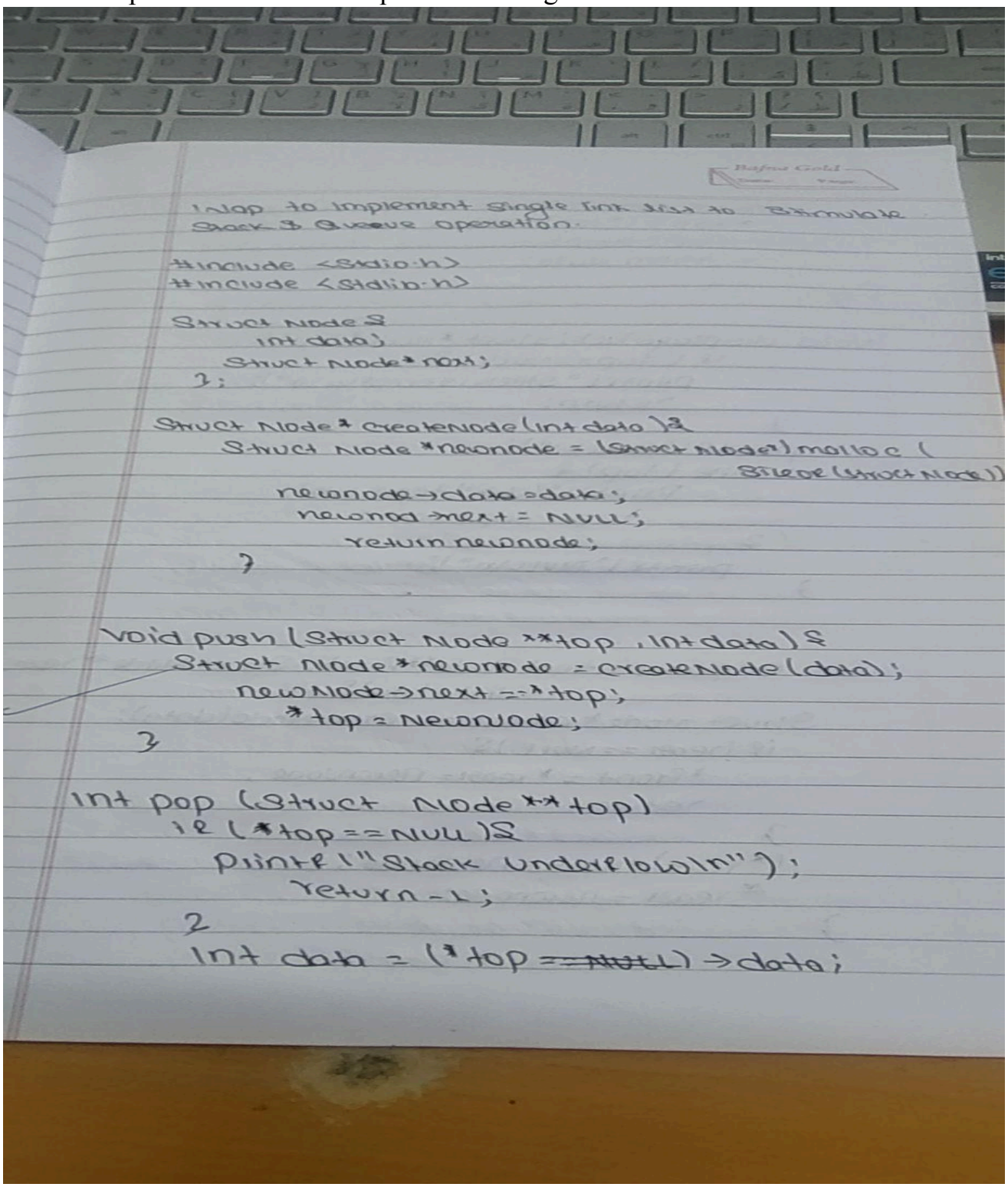
```
5 6 7

Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
6. Print the first linked list
7. Print the second linked list
8. Exit
Enter your choice: 3
The first linked list has been sorted.

Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
6. Print the first linked list
7. Print the second linked list
8. Exit
Enter your choice: 6
First Linked List: 1 -> 2 -> 4 -> NULL
```

```
Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
6. Print the first linked list
7. Print the second linked list
8. Exit
Enter your choice: 4
The first linked list has been reversed.

Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists

Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
2. Append elements to the second linked list
3. Sort the first linked list
```

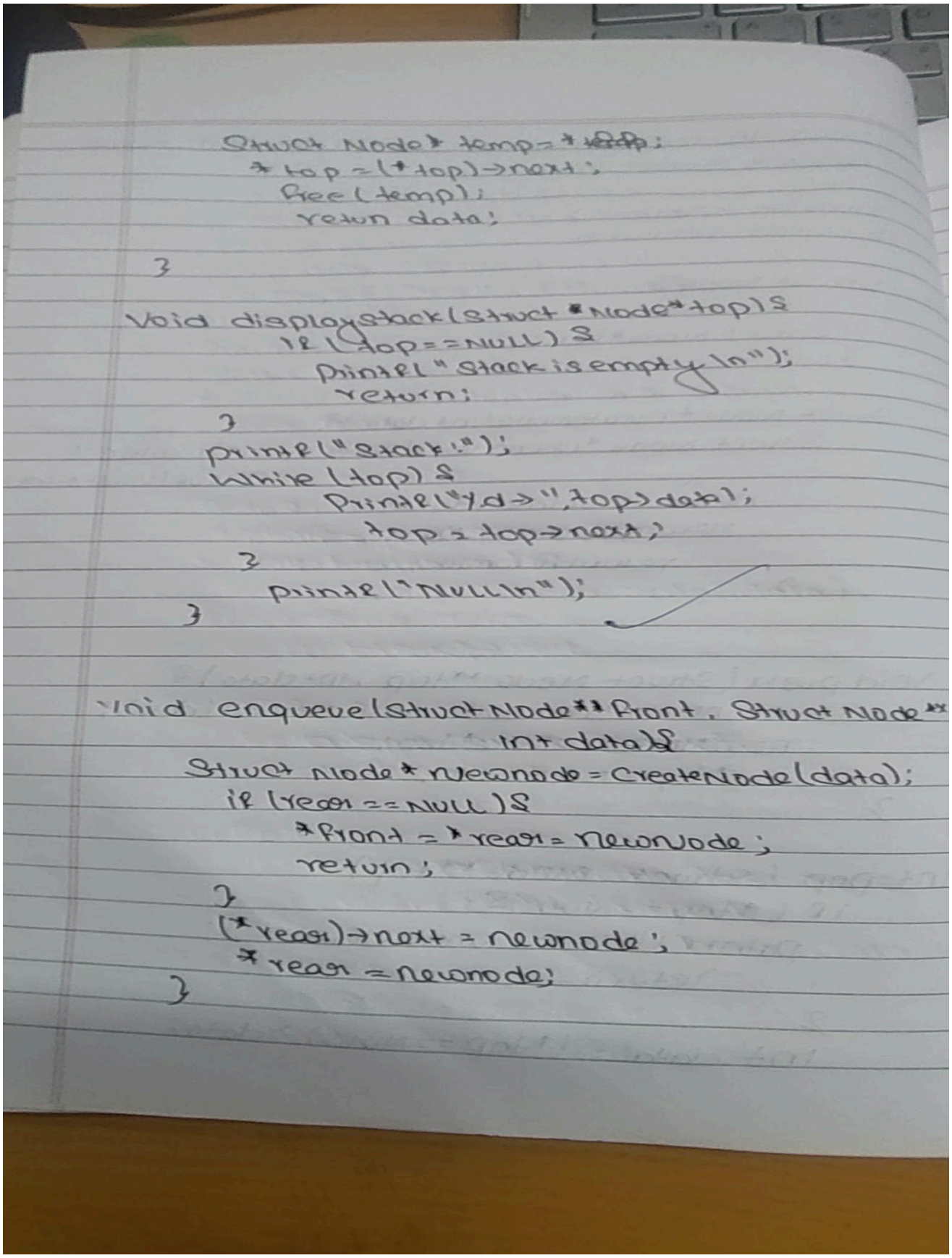
```
2. Append elements to the second linked list
Menu:
1. Append elements to the first linked list
2. Append elements to the second linked list
1. Append elements to the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
2. Append elements to the second linked list
3. Sort the first linked list
3. Sort the first linked list
4. Reverse the first linked list
5. Concatenate the two linked lists
6. Print the first linked list
7. Print the second linked list
8. Exit
Enter your choice: 5
The two linked lists have been concatenated.
```

program 7

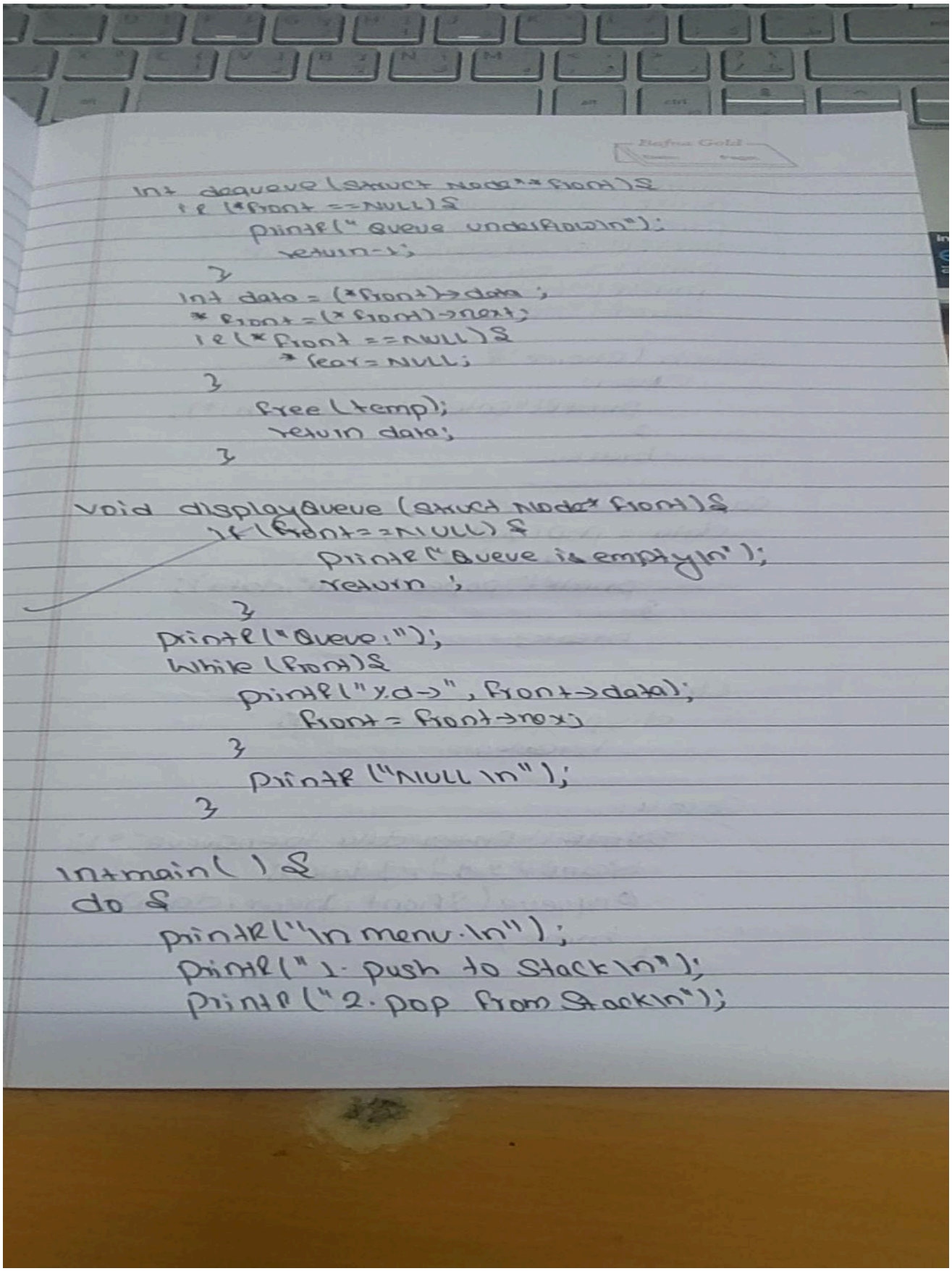
WAP to Implement to stack and queue from single linked list



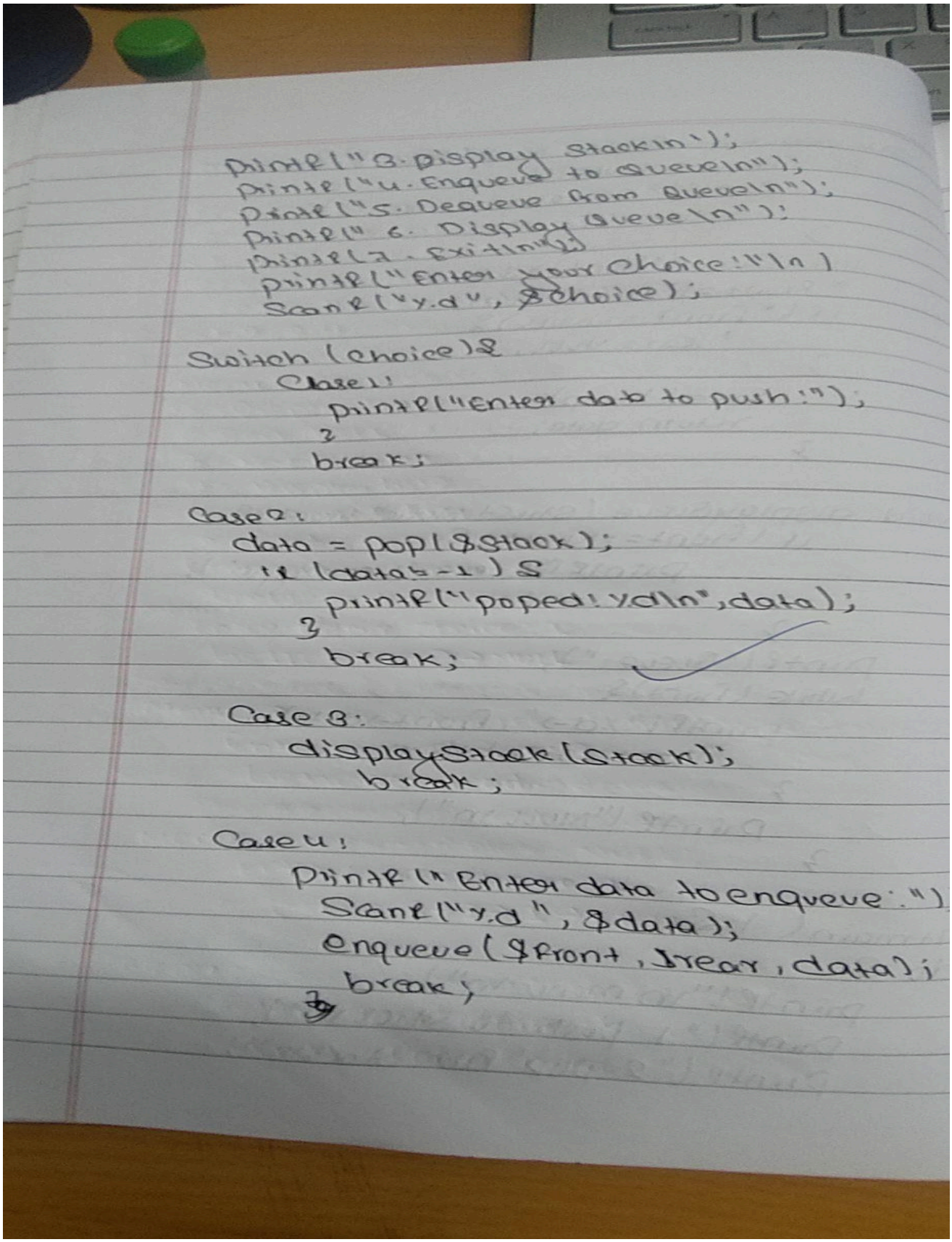
```
WAP to implement single link list to simulate
stack & queue operation.

#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 == NULL) -> data;
```



```
    struct Node* temp = *top;
    *top = (*top)->next;
    free(temp);
    return data;
}

void displayStack(struct Node* top){
    if (top == NULL){
        printf("Stack is empty\n");
        return;
    }
    printf("Stack:");
    while (top){
        printf("%d->", top->data);
        top = top->next;
    }
    printf("NULL\n");
}

void enqueue(struct Node** front, struct Node**
             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;
  *front = (*front)->next;
  if(*front == NULL){
      *rear = NULL;
  }
      free(temp);
      return data;
  }

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

int main(){
do{
    printf("\n menu.\n");
    printf("1. push to Stack\n");
    printf("2. pop from Stack\n");
```



```
printf("3. Display Stack\n");
printf("4. Enqueue to Queue\n");
printf("5. Dequeue from Queue\n");
printf("6. Display Queue\n");
printf("7. Exit\n");
printf("Enter your choice:\n");
scanf("%d", &choice);

switch (choice){
    case 1:
        printf("Enter data to push:");
    }
    break;

case 2:
    data = pop(&Stack);
    if (data != -1) {
        printf("poped: %d\n", data);
    }
    break;

case 3:
    displayStack(Stack);
    break;

case 4:
    printf("Enter data to enqueue:");
    scanf("%d", &data);
    enqueue(&front, &rear, data);
    break;
}
```

```
case 5:
    data = dequeue(&front);
    if (data!=-1)
        printf("Dequeued: %d\n", data);
        break;

    default:
case 6:
    display front
    displayQueue(front);
        break;

case 7:
    printf("Exiting ....\n");
        break;
    default:
        printf("Invalid choice. Try again.\n");
}
while (choice != 7);
    return 0;
}
```

List 1 : 3 → 2 → 1 → 1 → 2 → 3 → 4 → NULL.

~~Demo 1~~

output of program 2:

Choose an oper:

push(stack)
pop (stack)
Enqueue(Queue)
Dequeue(Queue)
exit

code:

```
#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;
   printf("%d pushed to stack\n", data);
}

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

void peek(struct Node* top) {
   if (top == NULL) {
      printf("Stack is empty.\n");
      return;
   }
   printf("Top element is %d\n", top->data);
```

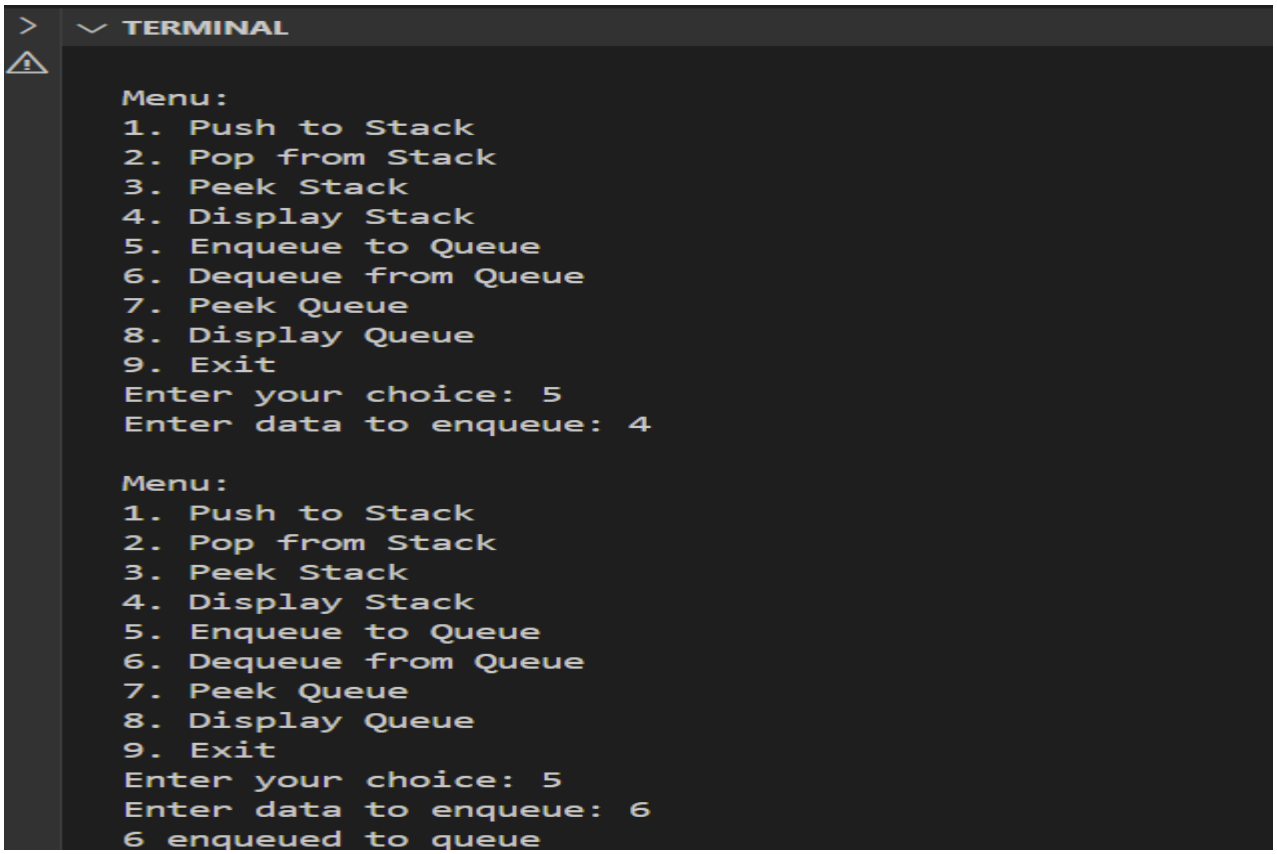
```
}

void displayStack(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    struct Node* temp = top;
    printf("Stack elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

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

void dequeue(struct Node** front) {
    if (*front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = *front;
    *front = (*front)->next;
    printf("%d dequeued from queue\n", temp->data);
    free(temp);
}

void peekQueue(struct Node* front) {
    if (front == NULL) {
```

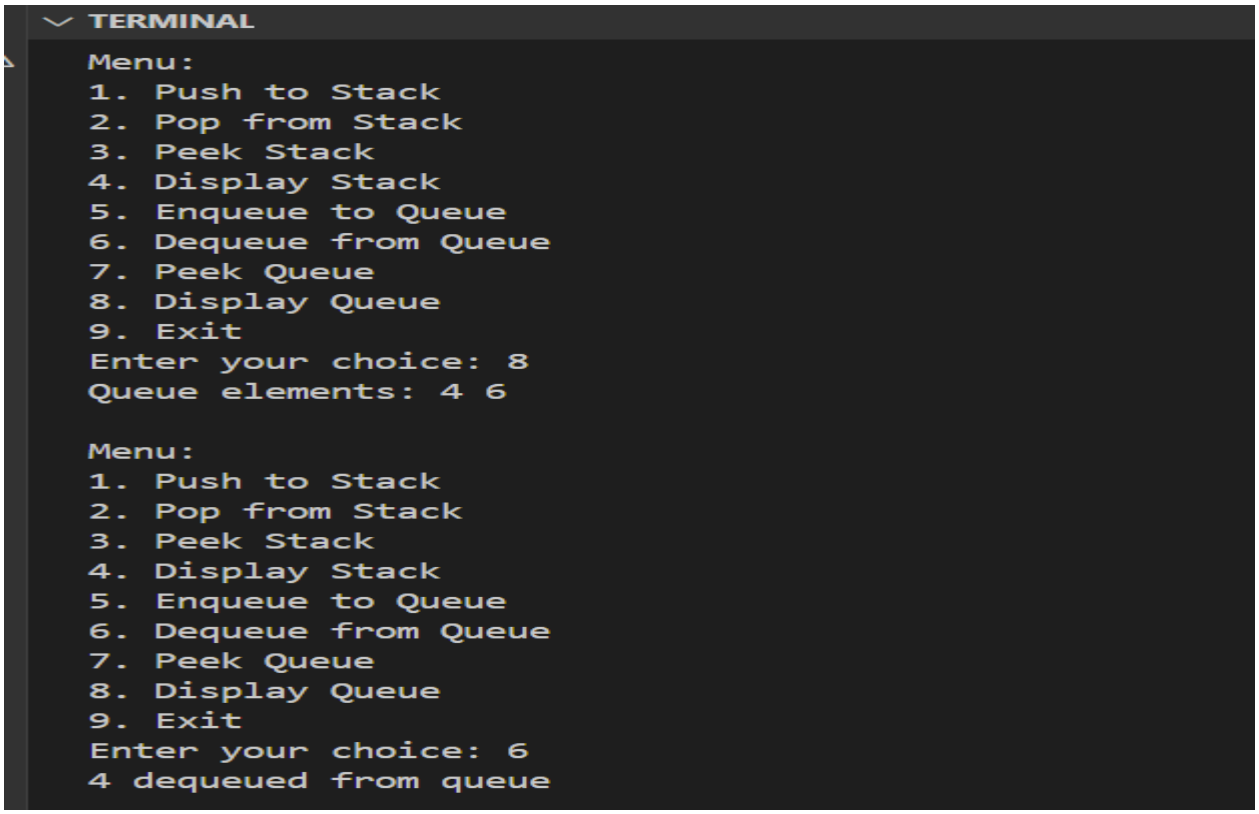
```
        printf("Queue is empty.\n");
        return;
    }
    printf("Front element is %d\n", front->data);
}

void displayQueue(struct Node* front) {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* stackTop = NULL;
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;
    int choice, data;

    do {
        printf("\nMenu:\n");
        printf("1. Push to Stack\n");
        printf("2. Pop from Stack\n");
        printf("3. Peek Stack\n");
        printf("4. Display Stack\n");
        printf("5. Enqueue to Queue\n");
        printf("6. Dequeue from Queue\n");
        printf("7. Peek Queue\n");
        printf("8. Display Queue\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                push(&stackTop, data);
                break;
            case 2:
                pop(&stackTop);
                break;
            case 3:
                peek(stackTop);
                break;
            case 4:
                displayStack(stackTop);
                break;
            case 5:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&queueFront, &queueRear, data);
                break;
            case 6:
                dequeue(&queueFront);
                break;
            case 7:
                peekQueue(queueFront);
                break;
            case 8:
                displayQueue(queueFront);
                break;
            case 9:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 9);

    return 0;
}
```

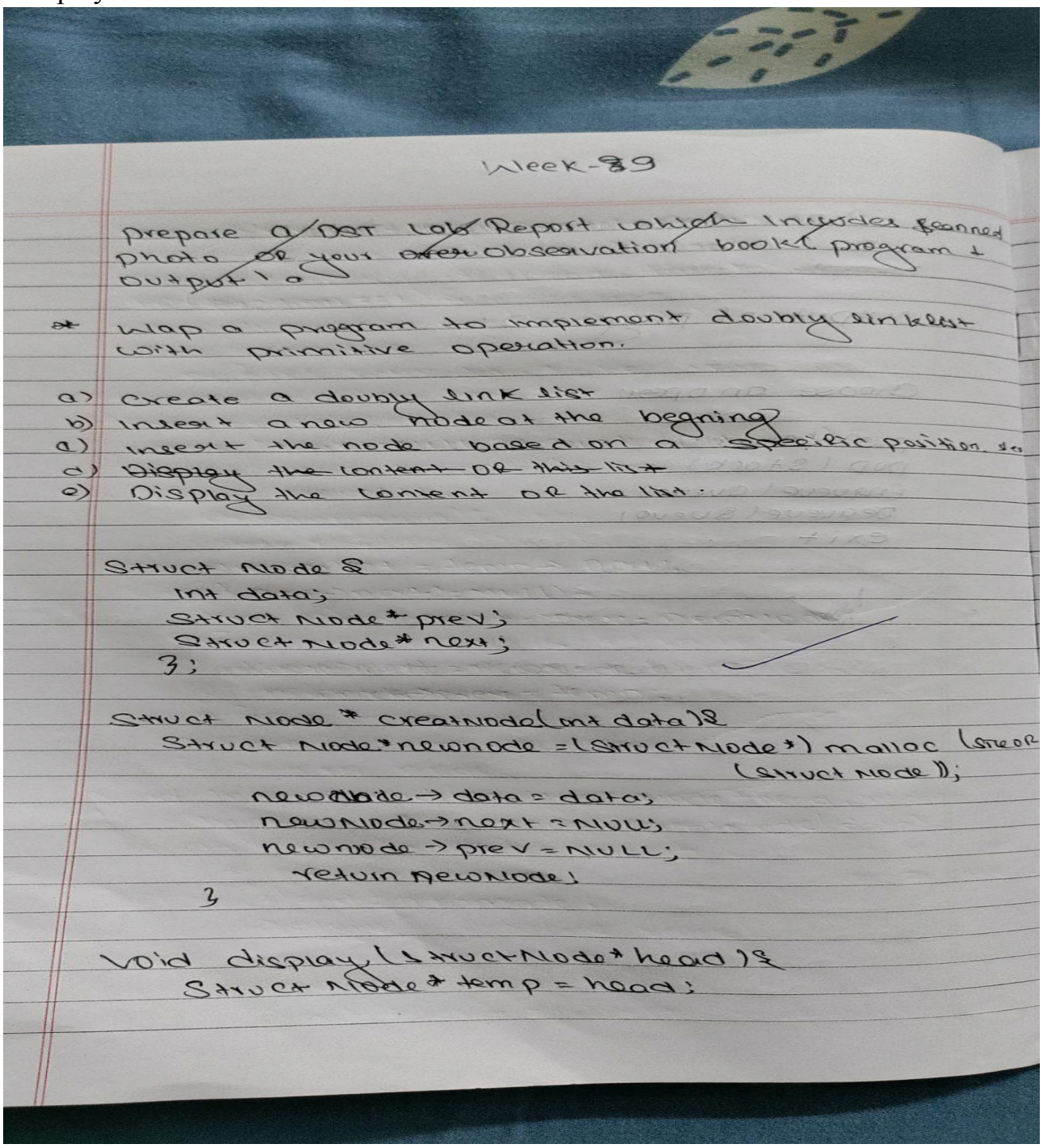
```
TERMINAL
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\Desktop\ds\"
e } ; if ($?) { .\tempCodeRunnerFile }

Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 1
Enter data to push: 4
4 pushed to stack

Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 1
Enter data to push: 3
3 pushed to stack
```

```
TERMINAL

Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 3
Top element is 3

Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 2
3 popped from stack
```


TERMINAL
Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 5
Enter data to enqueue: 4
Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 5
Enter data to enqueue: 6
6 enqueued to queue


TERMINAL
Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 8
Queue elements: 4 6
Menu:
1. Push to Stack
2. Pop from Stack
3. Peek Stack
4. Display Stack
5. Enqueue to Queue
6. Dequeue from Queue
7. Peek Queue
8. Display Queue
9. Exit
Enter your choice: 6
4 dequeued from queue

program 9
wap a program to implement doubly linklist with primitive operation.
1.create a doubly linked list
2.insert a new node at the beginning
3.insert the node base on specific position
4.display the contents of the list



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

void insertAtBeginning(struct Node*head, int data)
{
  struct Node* newNode = createNode(data);
  if (*head==NULL){
      *head = newNode;
      (*head)->prev = newNode;
      *head = newNode;
  } else{
      newNode->next = *head;
      (*head)->prev = newNode;
      *head = newNode;
  }
}

void insertAtposition(struct Node*head, int data, int pos
```

```
Struct Node* newNode = creatNode(data);
 struct Node * temp = *head;

if(position == 1){
    insertAtbegning(head, data);
     return;
  }

int count = 1;
while (temp != NULL && count < position - 1){
     temp = temp->next;
      count++;
   }

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


void insertAtend(struct Node *head, int data){
    Struct Node* newNode = creatNode(data);
    if(*head == NULL){
     {
      *head = newNode;
      return;
     }
```

```
Struct node* temp = *head;
While (temp->next != NULL)
        temp = temp->next;
  }
     temp->next = newNode;
    newNode->prev = temp;
  }

int main()
  Struct Node* head = NULL;
   int choice, data, position;

While(1)
  printf("\n Menu:\n");
   printf("1. Insert at beginning\n");
   printf("2. Insert at specific position\n");
   printf("3. Insert at end\n");
    printf("4. Display the list\n");
    printf("5. Exit\n");
    Printf("Enter the choice:");
    Scanf("%d", &choice:");

Switch(choice)
   Case 1:
     printf("Enter data to Insert at the begning:");
      Scanf("%d", &data);
       insertAtBeginning(&head, data);
        break;

  Case 2:
    Printf("Enter data to insert:");
     Scanf("%d", &data);
```

```
printf("Enter the position to insert at: ");
scanf("%d", &position);
insertAtPosition(&head, data, position);
break;

case 3:
printf("Enter data to be inserted at the end:\n");
scanf("%d", &data);
insertAtend(&head, data);
break;

case 4:
display(head);
break;

case 5:
exit(0);
default:
printf("Invalid choice. Plese try again\n");
}
return 0;
}
```

Output:

Menu!
1. Insert at the beginning
2. Insert at a specific location
3. Insert at the end
4. Display the list
5. Exit

Enter your choice: 1
Enter the data to insert at the begning: 2

Menu!
''

Enter your choice: 1
Enter the data at the begning: 3

Menu!!
''
Enter your choice: 4
Current list 2 ⇄ 3

Menu!

Enter your choice: 3
Enter the data to insert at end : 5

Enter your choice: 4
Current list 2 ⇄ 3 ⇄ 5

code:

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

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

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

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

   if (*head != NULL) {
      (*head)->prev = newNode;
   }

   *head = newNode;
}

void insertAtPosition(struct Node** head, int data, int position) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;

   if (position == 1) {
      insertAtBeginning(head, data);
      return;
```

```
    }

    struct Node* temp = *head;
    int count = 1;
    while (temp != NULL && count < position - 1) {
        temp = temp->next;
        count++;
    }

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

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

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }

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

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

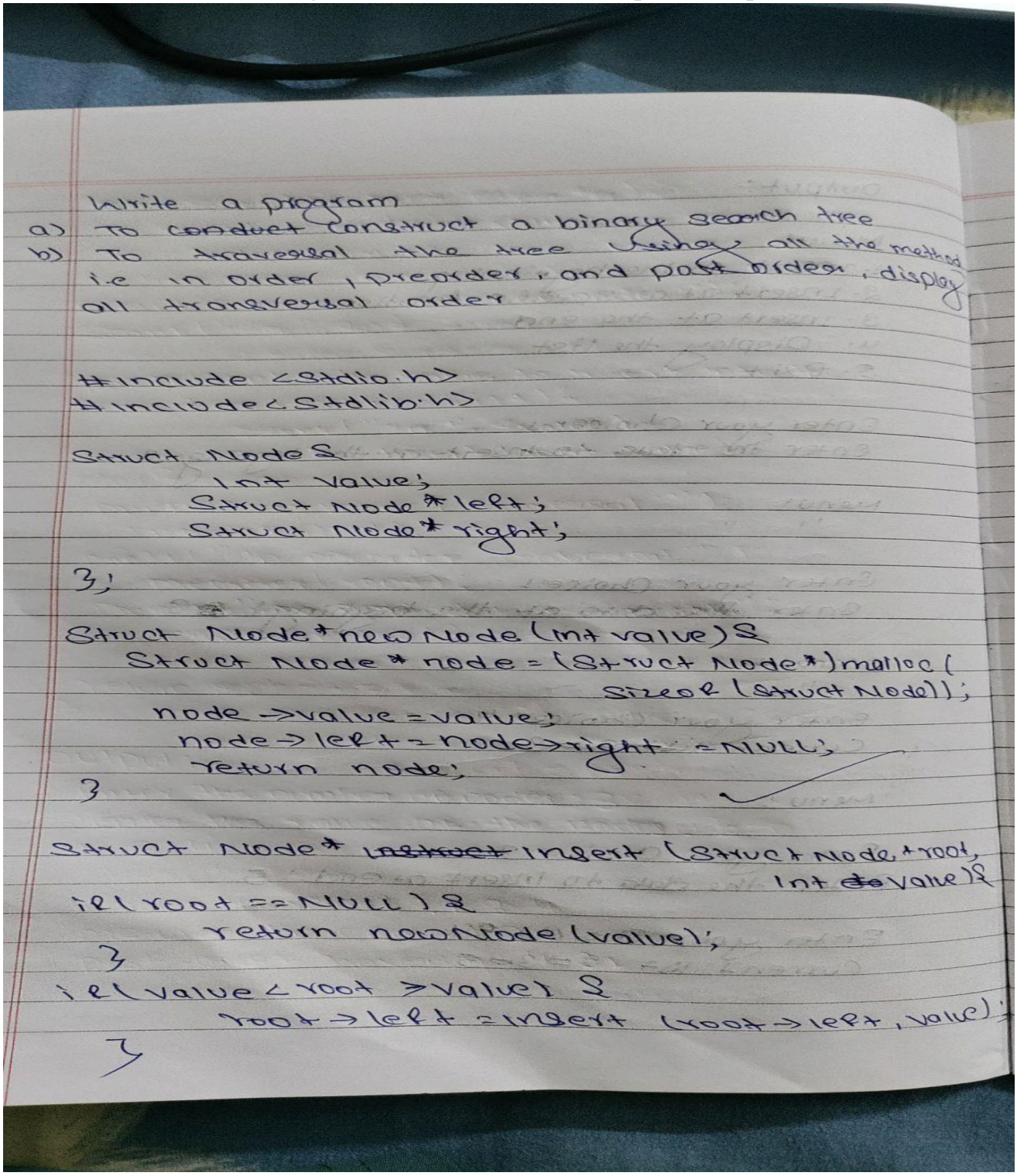
```
int choice, data, position;

do {
   printf("\nMenu:\n");
   printf("1. Create a Doubly Linked List\n");
   printf("2. Insert at the Beginning\n");
   printf("3. Insert at a Specific Position\n");
   printf("4. Display the List\n");
   printf("5. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

   switch (choice) {
      case 1:
         printf("Enter data to create the list: ");
         scanf("%d", &data);
         head = createList(data);
         printf("Doubly Linked List created with data %d\n", data);
         break;
      case 2:
         printf("Enter data to insert at the beginning: ");
         scanf("%d", &data);
         insertAtBeginning(&head, data);
         break;
      case 3:
         printf("Enter data to insert: ");
         scanf("%d", &data);
         printf("Enter position to insert the data: ");
         scanf("%d", &position);
         insertAtPosition(&head, data, position);
         break;
      case 4:
         displayList(head);
         break;
      case 5:
         printf("Exiting the program.\n");
         break;
      default:
         printf("Invalid choice. Please try again.\n");
   }
```

```
    } while (choice != 5);

    return 0;
}
```

```
TERMINAL
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\Desktop\ds\" ;
e } ; if ($?) { .\tempCodeRunnerFile }

Menu:
1. Create a Doubly Linked List
2. Insert at the Beginning
3. Insert at a Specific Position
4. Display the List
5. Exit
Enter your choice: 1
Enter data to create the list: 3
Doubly Linked List created with data 3

Menu:
1. Create a Doubly Linked List
2. Insert at the Beginning
3. Insert at a Specific Position
4. Display the List
5. Exit
Enter your choice: 2
Enter data to insert at the beginning: 4

Menu:
1. Create a Doubly Linked List
2. Insert at the Beginning
3. Insert at a Specific Position
4. Display the List
5. Exit
Enter your choice: 2
Enter data to insert at the beginning: 7
```

```
TERMINAL

Menu:
1. Create a Doubly Linked List
2. Insert at the Beginning
3. Insert at a Specific Position
4. Display the List
5. Exit
Enter your choice: 4
List contents: 7 4 3

Menu:
1. Create a Doubly Linked List
2. Insert at the Beginning
3. Insert at a Specific Position
4. Display the List
5. Exit
Enter your choice: 3
Enter data to insert: 5
Enter position to insert the data: 2

Menu:
1. Create a Doubly Linked List
2. Insert at the Beginning
3. Insert at a Specific Position
4. Display the List
5. Exit
Enter your choice: 4
List contents: 7 5 4 3
```

program 9
wap a program
1.To conduct construct a binary search tree
2.To traversal the tree using all the method i.e inorder ,preorder ,postorder



```
Write a program
a) To conduct construct a binary search tree
b) To traversal the tree using all the method
   i.e in order, preorder, and post order, display
   all transversal order

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

Struct Node{
      int value;
      Struct Node *left;
      Struct Node* right;
};

Struct Node* newNode(int value){
   Struct Node * node = (Struct Node*)malloc(
                        sizeof(Struct Node));
    node->value = value;
     node->left = node->right = NULL;
     return node;
}

Struct Node* ~~Struct~~ Insert (Struct Node *root,
                                int ~~de~~ value){
if(root == NULL){
      return newNode(value);
}
if(value < root >value){
      root->left = Insert (root->left, value);
}
```

```
else {
    root->right = insert(root->right, value);
  }
  return root;
}

void inorder (struct Node* root) {

  if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->value);
        preorder(root->left);
        preorder(root->right);
  }
}


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


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

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

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

printf("Enter the value to insert into the BST:\n");
for (int i=0; i<n; i++){
        scanf("%d", &value);
        root = insert(root, value);
    }
printf("\n Inorder Traversal:");
    inorder(root);
printf("\n Preorder Traversal:");
    preorder(root);
printf("\n Postoder Traversal:");
    postoder(root);

return 0;
}
```

Output:

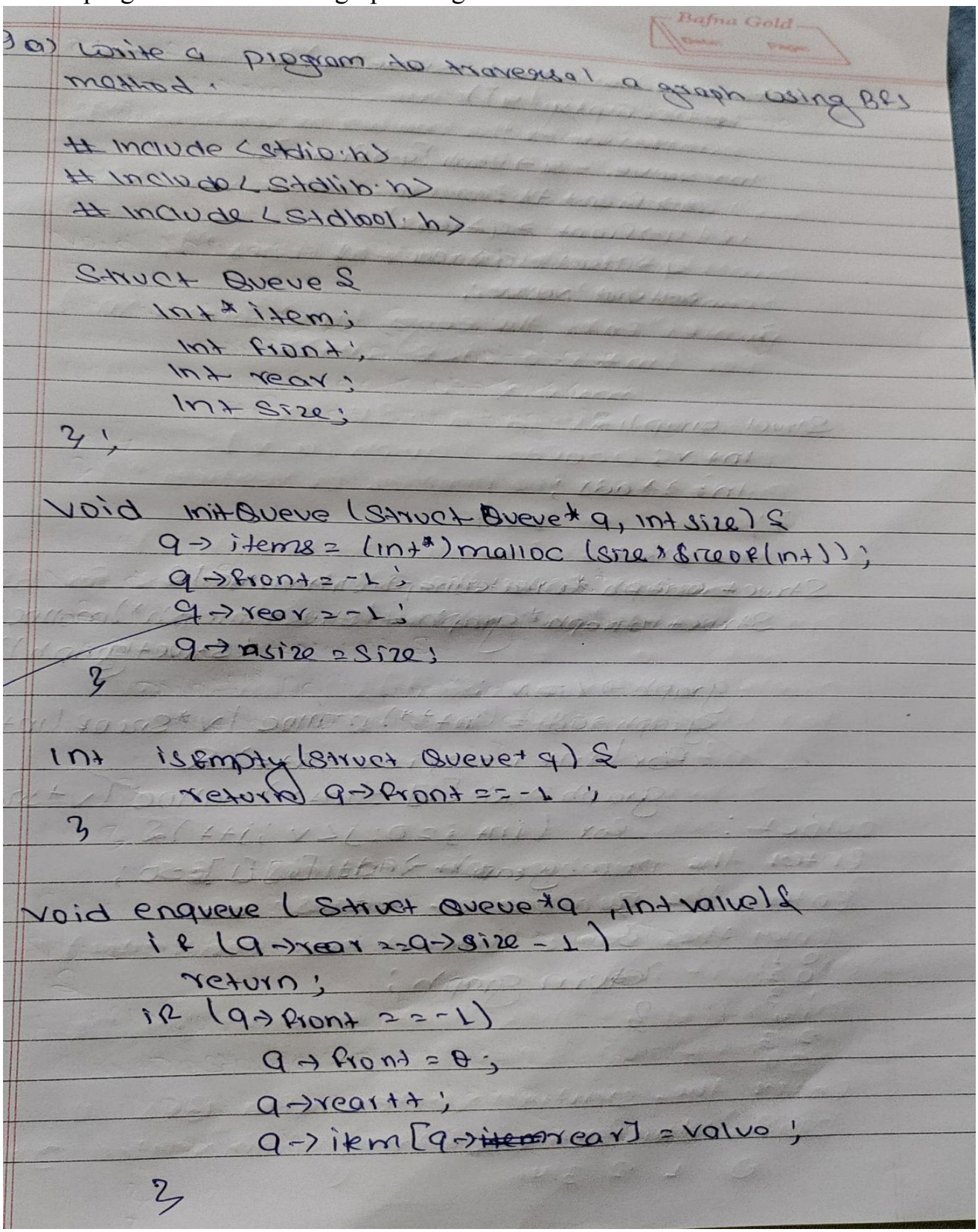
```
Enter the number of nodes: 3
Enter the value of the nodes:
1 3 4

Inorder traversal: 1 3 4
pre-order traversal: 1 3 4
post-order traversal: 4 3 1
```

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

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

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

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

    printf("Enter the values to insert in the BST:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }
    printf("\nInorder Traversal: ");
    inorder(root);
    printf("\nPreorder Traversal: ");
    preorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);
    return 0;
}
```

```
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\Desktop\ds\" ;
e } ; if ($?) { .\tempCodeRunnerFile }
Enter the number of nodes: 4
Enter the values to insert in the BST:
1 2 3 5

Inorder Traversal: 1 2 3 5
Preorder Traversal: 1 2 3 5
Postorder Traversal: 5 3 2 1
PS C:\Users\Soham\OneDrive\Desktop\ds>
```

program 9
write a program to traversal a graph using BFs methods



3 a) Write a program to traversal a graph using BFS method.

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

Struct Queue {
    int* item;
    int front;
    int rear;
    int size;
};

Void initQueue (Struct Queue* q, int size) {
    q -> items = (int*) malloc (size * sizeof(int));
    q -> front = -1;
    q -> rear = -1;
    q -> size = size;
}

int isEmpty (Struct Queue* q) {
    return q -> front == -1;
}

Void enqueue ( Struct Queue *q , int value) {
    if (q -> rear == q -> size - 1)
        return;
    if (q -> front == -1)
        q -> front = 0;
    q -> rear++;
    q -> item [q -> rear] = value;
}
```

```
int dequeue (struct Queue*q){
    if(isempty(q))
        return -1;
    int value = q->item[q->front];
    q->front++;
    if(q->front > q->rear)
        q->front = q->rear = -1
    return value;
}

struct graph{
    int v;
    int **adj;
};

Struct Graph *creategraph(int value){
    Struct Graph* graph = (struct graph*)malloc(sizeof
                          (struct graph));
    graph->v = v;
    graph->adj = (int**) malloc (v*sizeof(int*));
    for(int i=0; i<v; i++){
        graph->adj[i] =(int*)malloc (v*sizeof(int)
        for (int j=0; j<v; j++){
            graph->adj[i][j] = 0;
        }
    }
    return graph;
}
```

```
void addEdge(struct Graph* graph, int u, int v) {
    graph->adj[u][v] = 1;
    graph->adj[v][u] = 1;
}

void bfs(struct Graph* graph, int start) {
    bool* visited = (bool*) malloc(graph->v * sizeof(bool));
    for(int i=0; i<graph->v; i++) {
        visited[i] = false;
    }

Struct queue q;
    initQueue(&q, graph->v);
    visited[started] = true;
    enqueue(&q, start);

    printf("BFS Traversal starting from vertex %d: ", start);

while(!isEmpty(&q)) {
        int node = dequeue(&q);
        printf("%d ", node);
        for(int i=0; i<graph->v; i++) {
        if(graph->adj[node][i]==1 && !visited[i]) {
            visited[i] = true;
            enqueue(&q, i);
    }
    }
    }
    printf("\n");
}
```

```
int main() {
    int v, e;
    printf("Enter the number of vertices: ");
    scanf("%d", &v);
    struct Graph* graph = createGraph(v);

    printf("Enter the number of edges: ");
    scanf("%d", &e);
    printf("Enter the edges (u v) format:\n");
    for (int i = 0; i < e; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        addEdge(graph, u, v);
    }

    int start;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &start);

    bfs(graph, start);
    return 0;
}
```

Output:

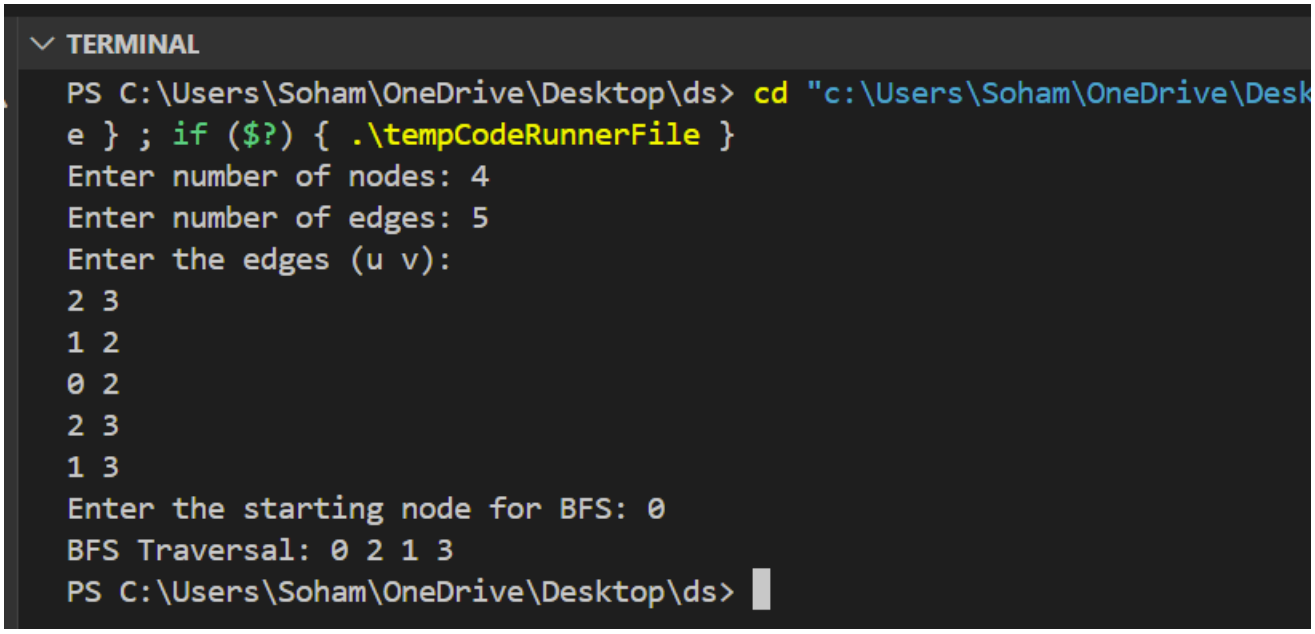
```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (start and end vertex):
0 1
0 2
1 3
1 4
2 3
3 4
Enter the starting vertex for BFS: 0
BFS traversal starting from vertex 0:
0 1 2 3 4
```

code:

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

#define MAX 100

struct Queue {
   int items[MAX];
   int front, rear;
};

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

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

void enqueue(struct Queue* q, int value) {
   if (q->rear == MAX - 1) {
      printf("Queue is full\n");
   } else {
      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;
   } else {
      int item = q->items[q->front];
      if (q->front == q->rear) {
         q->front = q->rear = -1;
```

```
        } else {
            q->front++;
        }
        return item;
    }
}

void bfs(int graph[MAX][MAX], int n, int start) {
    struct Queue q;
    initQueue(&q);
    int visited[MAX] = {0};

    visited[start] = 1;
    enqueue(&q, start);

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

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

int main() {
    int n, e, u, v;

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

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

```
        graph[u][v] = 1;
        graph[v][u] = 1;
    }

    int start;
    printf("Enter the starting node for BFS: ");
    scanf("%d", &start);

    printf("BFS Traversal: ");
    bfs(graph, n, start);

    return 0;
}
```



program 9
write a program to traversal a graph using DFs method

write a program to traverse a graph using DFS method;

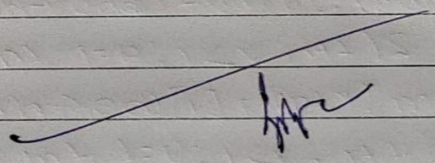
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Graph {
    int V;
    int **adj;
};

struct Graph* creatGraph(int v) {
    struct Graph *graph = (struct Graph*) malloc (size o
                                                  (struct Gr
    graph->v = v;
    graph->adj = (int**) malloc(v*sizeof(int* ));
    for(int i=0; i<v; i++) {
    graph->adj[i] = (int*) malloc(v*sizeof(int));
    for(int j=0; i<v; j++) {
        graph->adj[i][j] = 0;
    }
    }
    return graph;

}

void addEdge (struct Graph* graph, int u, int v) {

    graph->adj[u][v]=1;
    graph->adj[v][u] = 1;
}
```

```
void dfs (struct Graph*graph , int node , bool* visited)
    visited[node] = true;
    printf("%d",node);

for (int i=0 ; i<graph->v ;i++)
    if (graph->adj[node][i]==1 && !visited[i])
        dfs(graph ,i ,visited);
    }
  }
}

int main()
  int v, e;
  printf("Enter the number of vertices:");
   scanf("%d",&v);
   struct Graph * graph = creatGraph(v);

printf("Enter the number of edges:");
 scanf("%d",&e);
 printf("Enter the edges (u v) Format:\n");
 for (int i=0 ; i<e ;i++)
     int u,v;
     scanf("%d %d", &u ,&v);
      addedge (graph, u, v);

}

int start;
 printf("Enter the starting vertex for DFS:");
  scanf("%d", &start);

bool *visited = (bool*) malloc (v* sizeof(bool));
```

```
for (int i=0; i<v; i++) {
    visited[i] = false;
}

printf("DFS traversal starting from vertex %d:", start);
dfs(graph, start, visited);
printf("\n");
return 0;
}
```

Output:

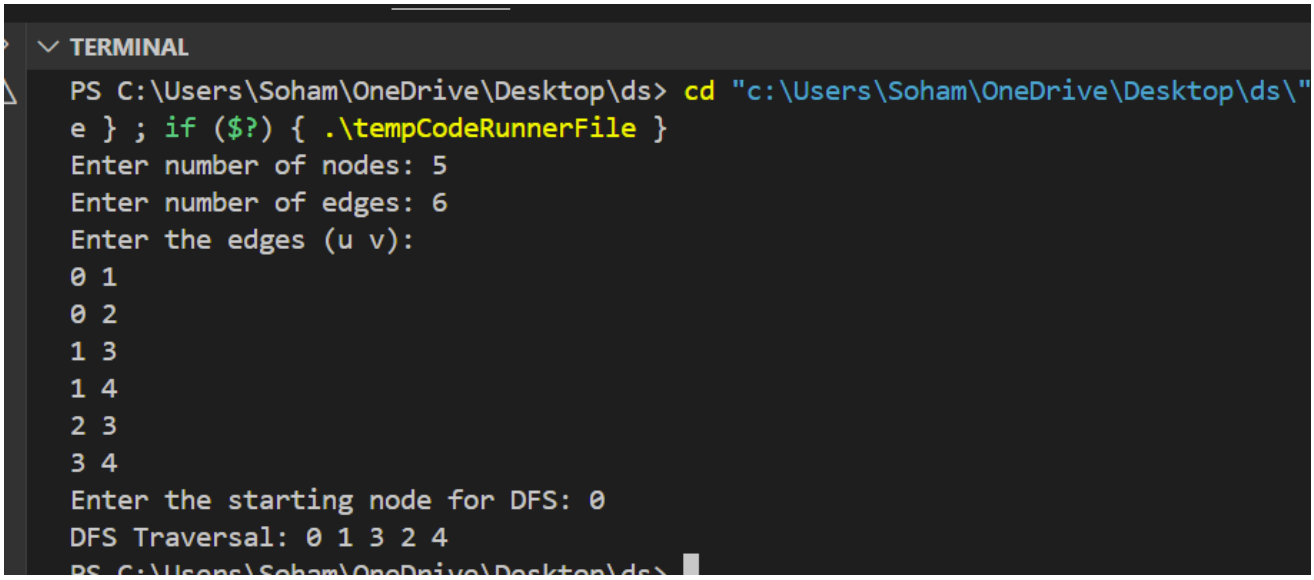
```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (start and end vertex):
0 1
0 2
0 3
1 4
2 3
3 4
Enter the starting vertex for DFS: 0

DFS traversal starting from vertex 0
0 1 3 2 4
```



code:

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

#define MAX 100

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

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

int main() {
   int n, e, u, v;

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

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

   int visited[MAX] = {0};
   int start;
   printf("Enter the starting node for DFS: ");
   scanf("%d", &start);

   printf("DFS Traversal: ");
   dfs(graph, n, start, visited);
```

```
    return 0;
}
```


TERMINAL
PS C:\Users\Soham\OneDrive\Desktop\ds> cd "c:\Users\Soham\OneDrive\Desktop\ds\"
e } ; if ($?) { .\tempCodeRunnerFile }
Enter number of nodes: 5
Enter number of edges: 6
Enter the edges (u v):
0 1
0 2
1 3
1 4
2 3
3 4
Enter the starting node for DFS: 0
DFS Traversal: 0 1 3 2 4