**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Data Structures using C Lab**

**(23CS3PCDST)**

***Submitted by***

**Shubhanshu Raj (1BM23CS325)**

***in partial fulfilment 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 **Shubhanshu Raj (1BM23CS325),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures using C Lab(23CS3PCDST) work prescribed for the said degree.

|  |  |
| --- | --- |
| Geetha N  Assistant Professor  Department of CSE, BMSCE | Dr. Kavitha Sooda  Professor &HOD  Department of CSE, BMSCE |

**Index**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** |
| 1 | 30/09/24 | Stack implementation using arrays |
| 2 | 7/10/24 | Infix to postfix conversion |
| 3 | 14/10/24 | Queue implementation using arrays |
| 4 | 21/10/24 | Circular queue implementation |
| 5 | 28/10/24 | Insertion operation in singly linked list |
| 6 | 11/11/24 | Deletion in Singly Linked List |
| 7 | 02/12/24 | Multiple operations on Linked list |
| 8 | 23/12/24 | Insertion operation using doubly linked list |
| 9 | 23/12/24 | Binary tree implementation |
| 10 | 23/12/24 | Graphs |

**Program 1**

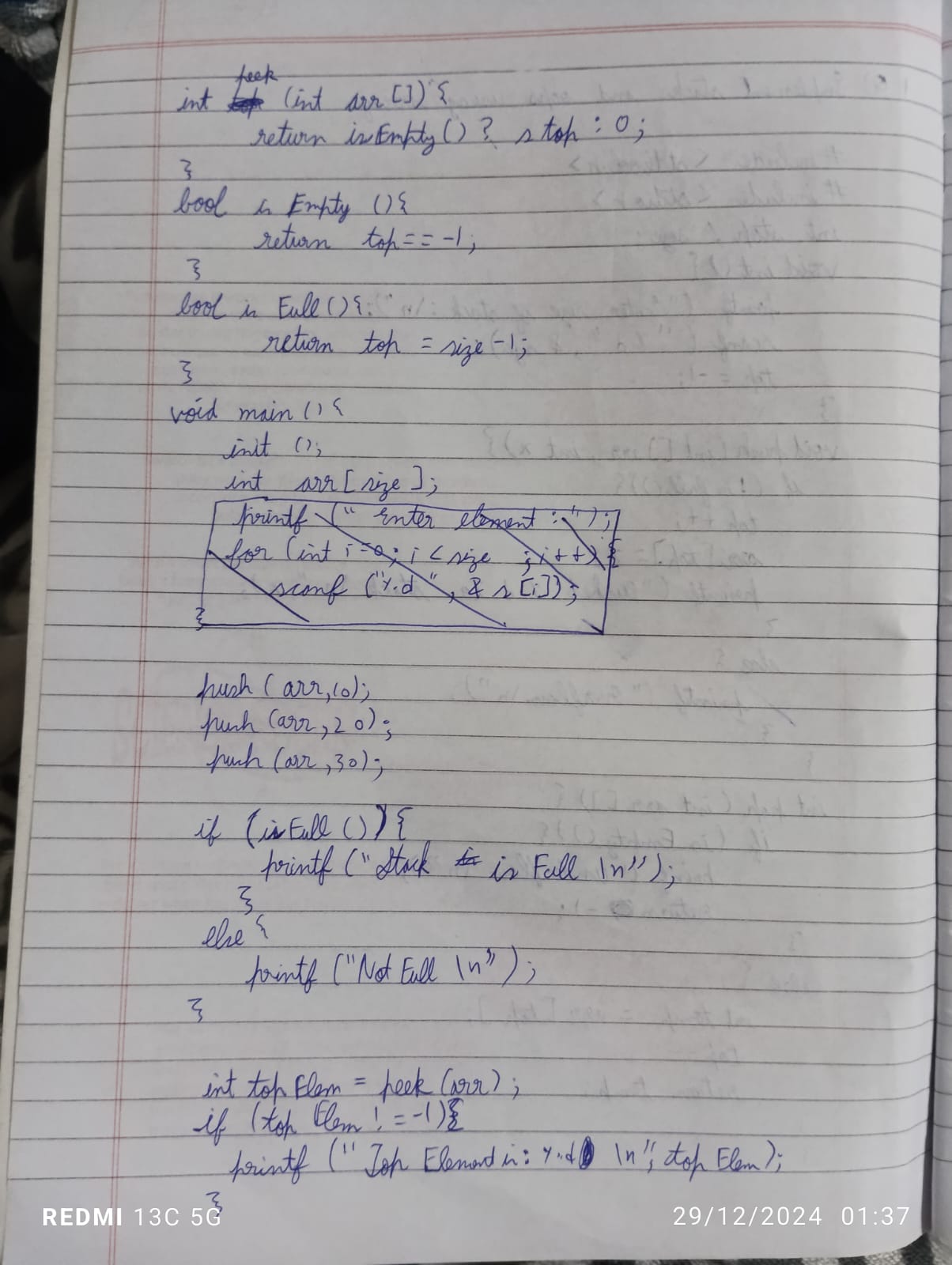
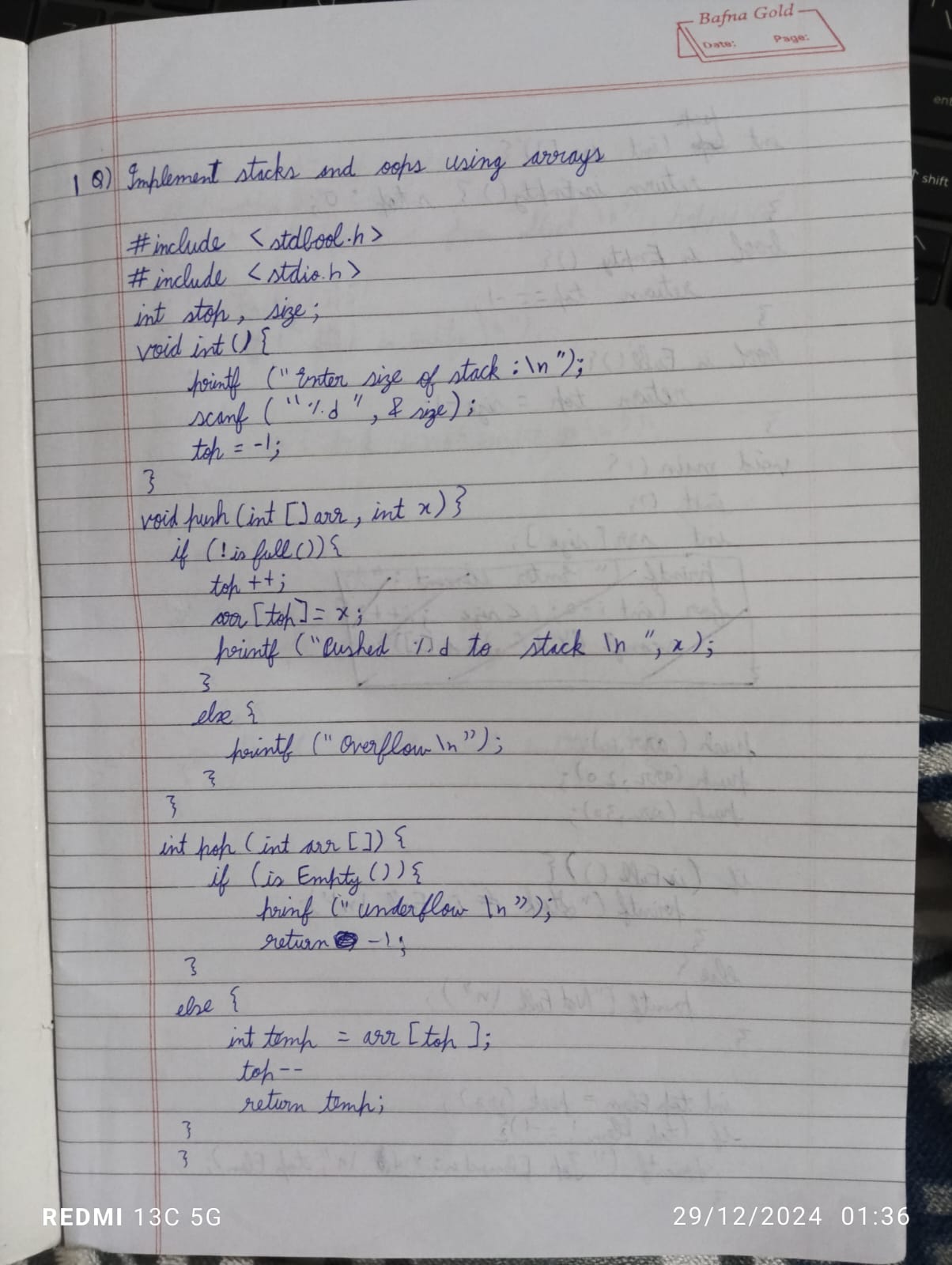
Write a program to simulate the working of stack using an array with the following:

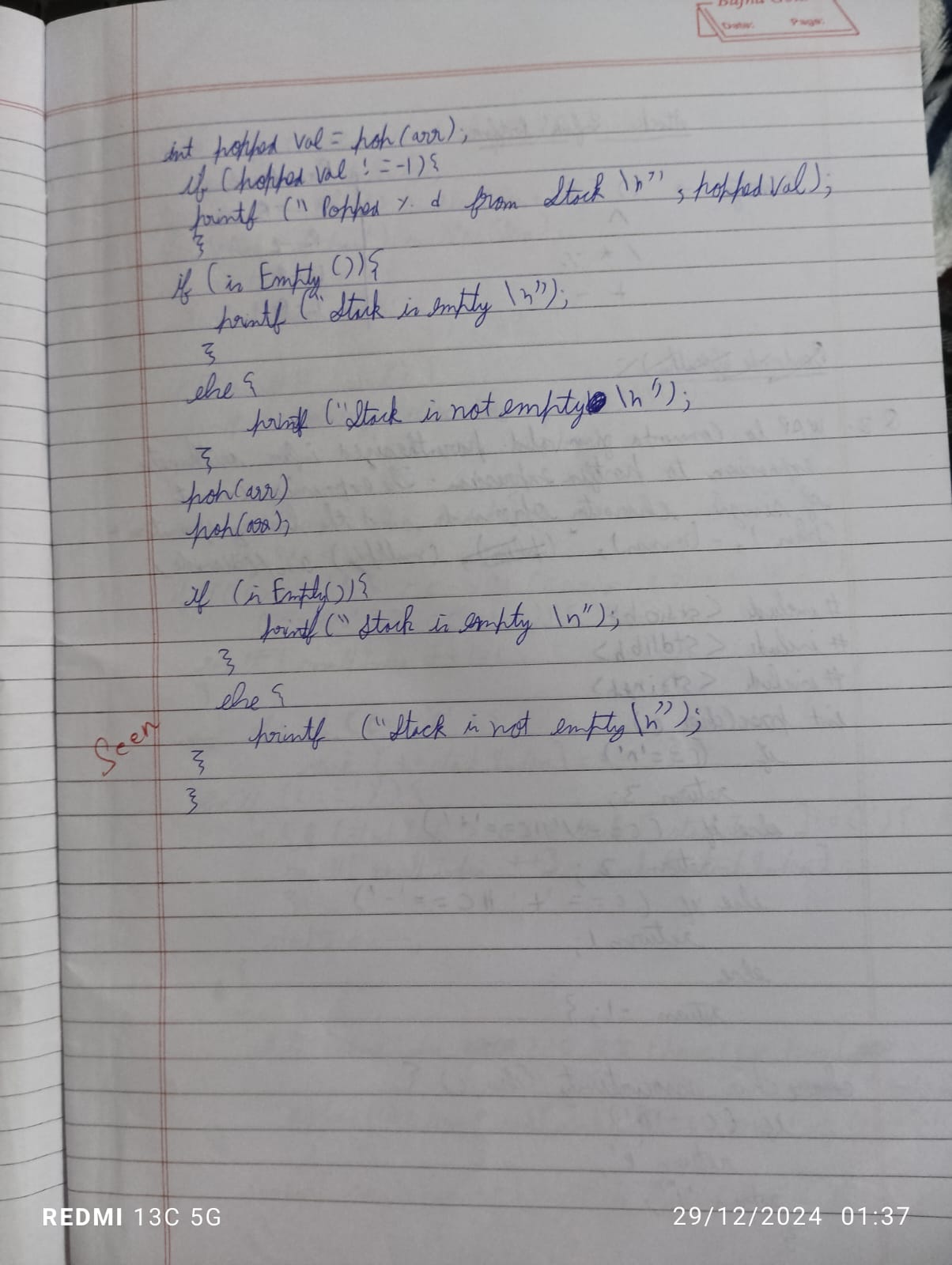
a) Push

b) Pop

c) Display

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



Code:

#include <stdio.h>

#include <stdlib.h>

#define SIZE 3

void push(int value);

void pop();

void display();

int stack[SIZE];

int top = -1;

int main() {

int value, choice;

while(1) {

printf("\nMenu\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice); // Corrected the format specifier

switch(choice) {

case 1:

printf("Enter the value to be inserted: ");

scanf("%d", &value);

push(value);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Invalid choice\n");

}

}

return 0;

}

void push(int value) {

if(top == SIZE - 1) {

printf("Stack is full\n");

} else {

top++;

stack[top] = value;

printf("Insertion success\n");

}

}

void pop() {

if(top == -1) {

printf("\nStack is empty\n");

} else {

printf("\nDeleted %d\n", stack[top]);

top--;

}

}

void display() {

if(top == -1) {

printf("\nStack is empty\n");

} else {

printf("Stack elements are:\n");

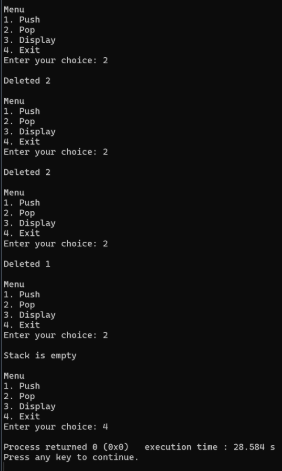
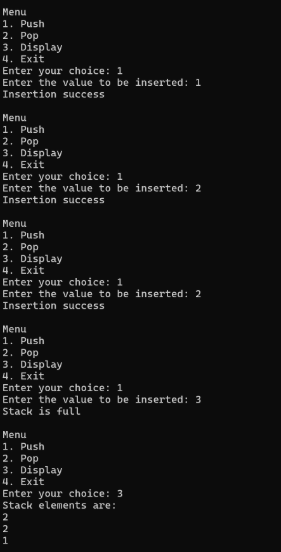
for(int i = top; i >= 0; i--) {

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

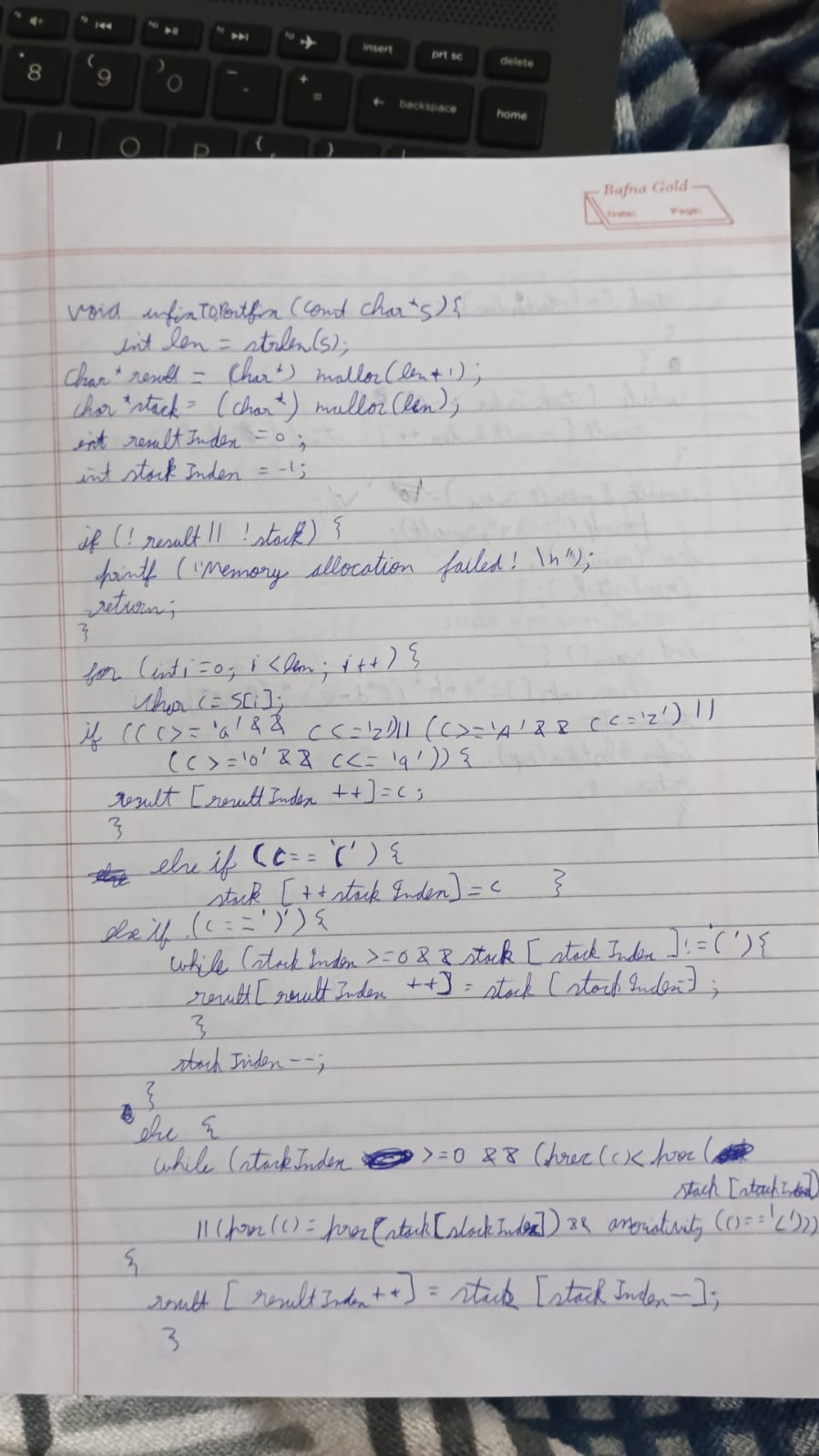
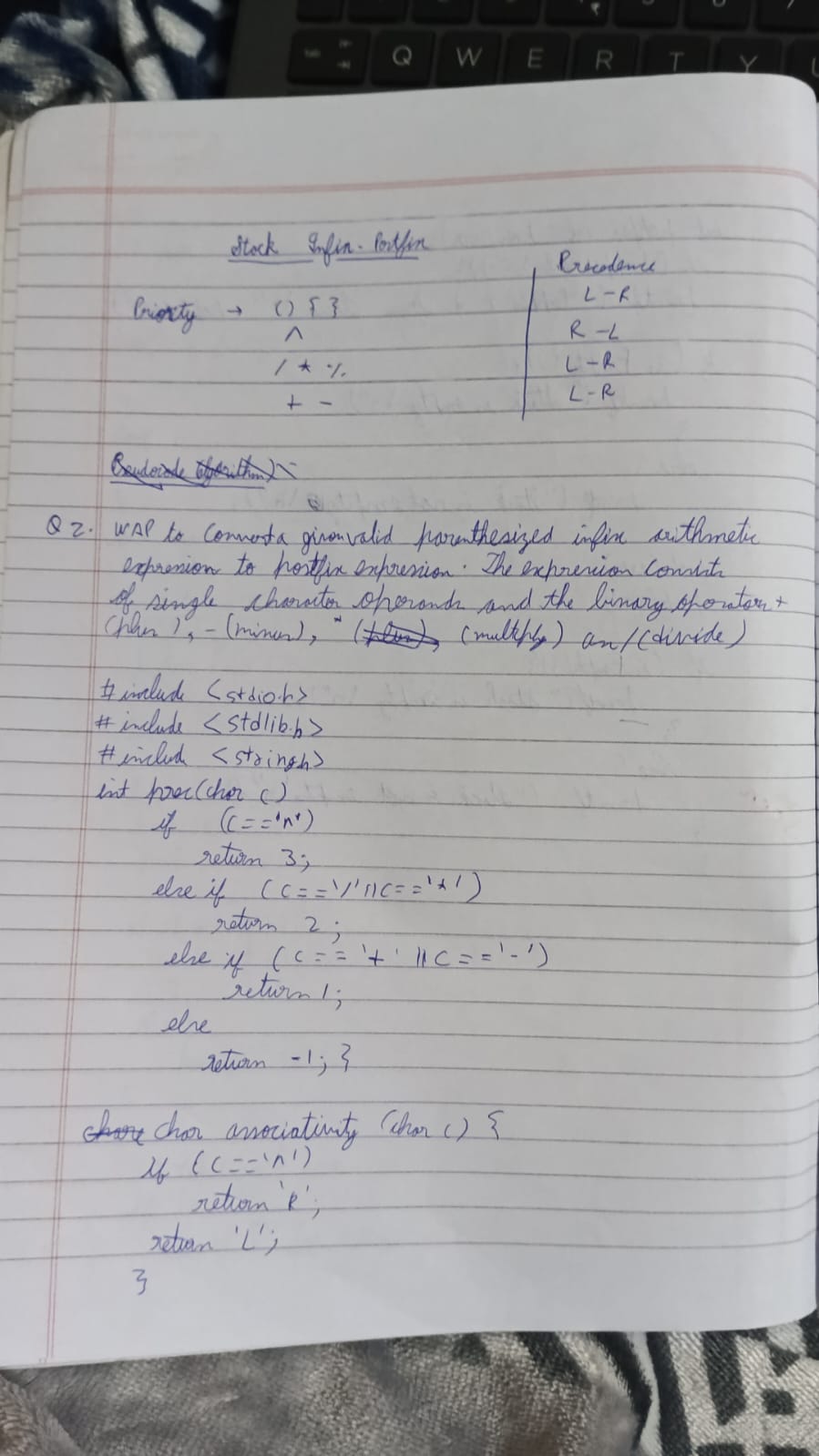
}

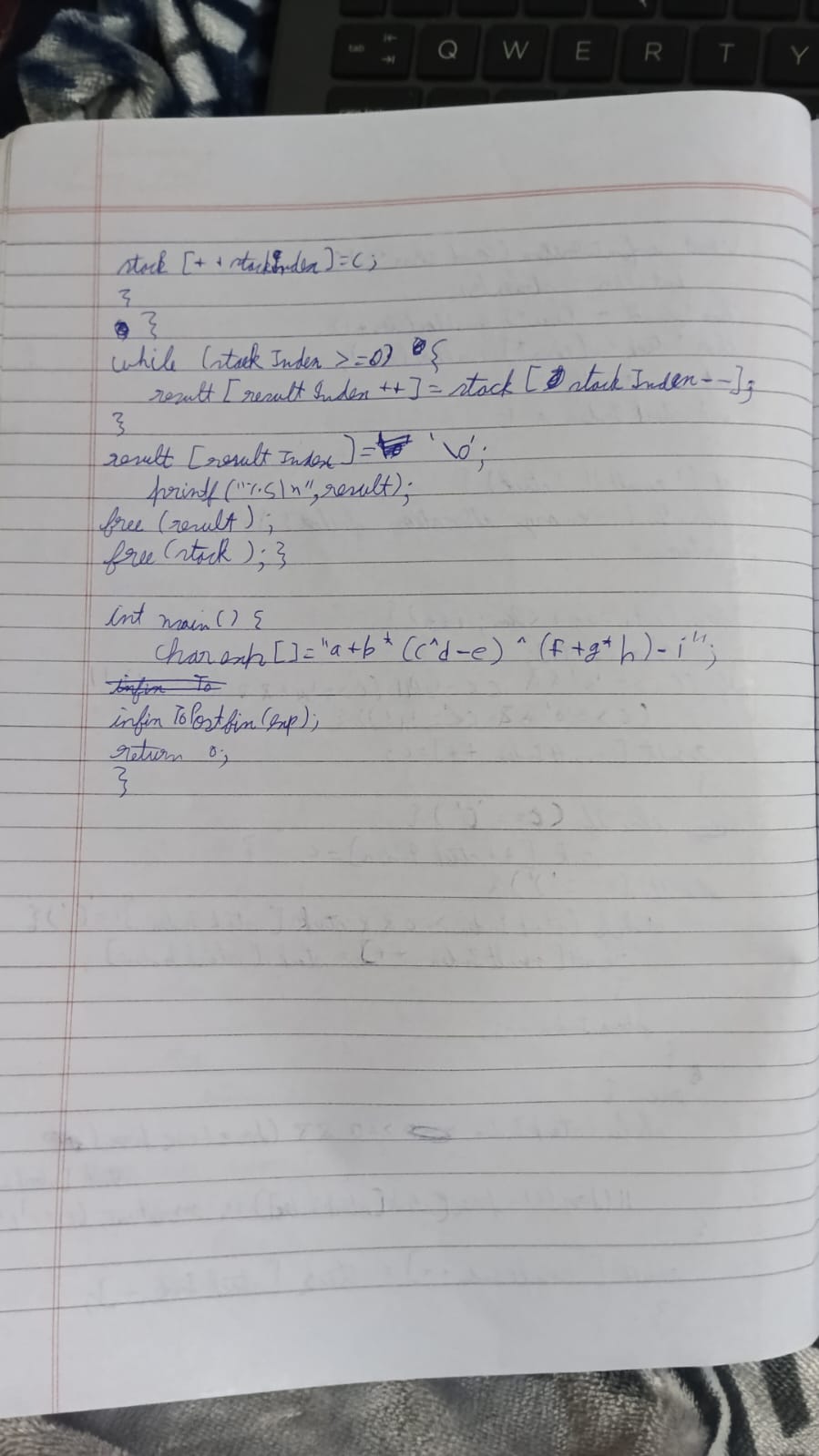
}

}



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





Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

int queue[MAX];

int front = -1, rear = -1;

int is\_empty() {

return front == -1 || front > rear;

}

int is\_full() {

return rear == MAX - 1;

}

void insert(int value) {

if (is\_full()) {

printf("Queue overflow! Cannot insert %d\n", value);

} else {

if (front == -1) {

front = 0;

}

queue[++rear] = value;

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

}

}

void delete() {

if (is\_empty()) {

printf("Queue underflow! Cannot delete\n");

} else {

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

front++;

if (is\_empty()) {

front = rear = -1;

}

}

}

void display() {

if (is\_empty()) {

printf("Queue is empty\n");

} else {

printf("Queue: ");

for (int i = front; i <= rear; i++) {

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

}

printf("\n");

}

}

int main() {

int choice, value;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter a value to insert: ");

scanf("%d", &value);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

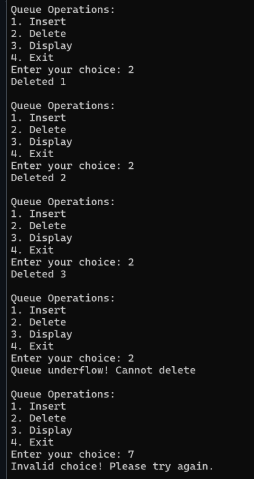
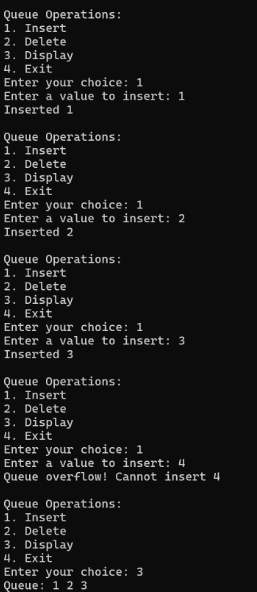
printf("Invalid choice! Please try again.\n");

}

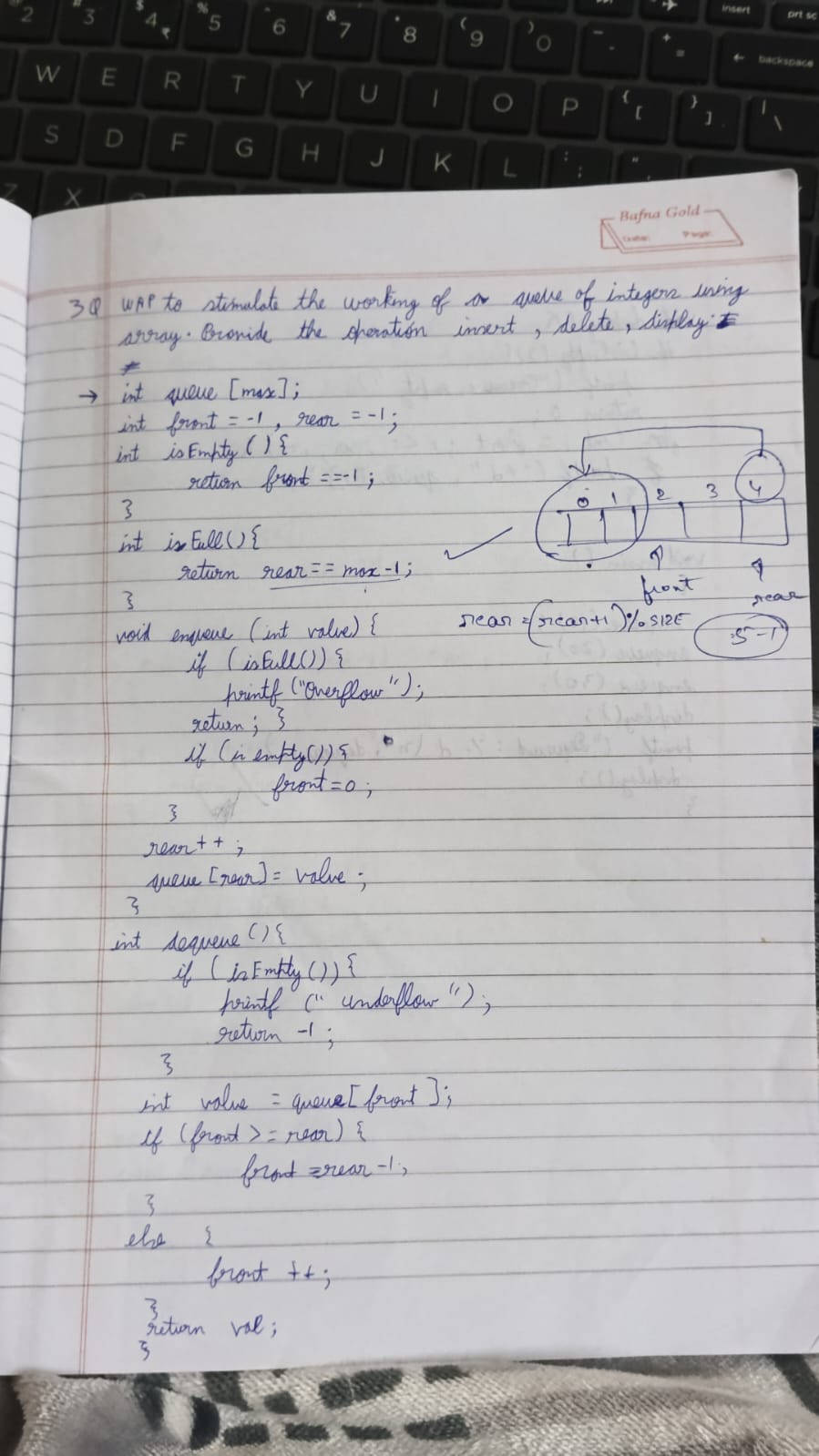
}

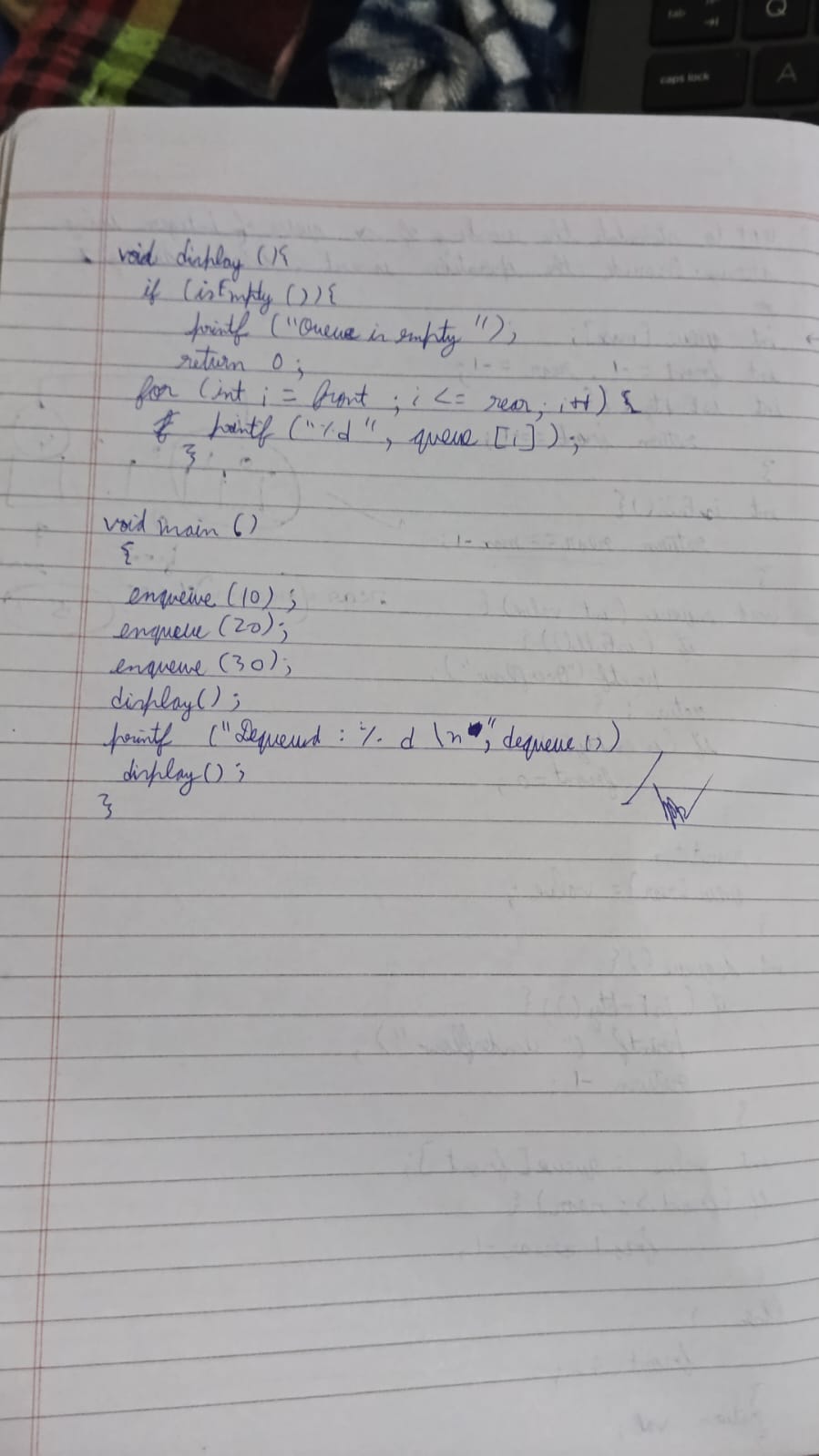
return 0;

}



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





Code:

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

int queue[MAX];

int front = -1, rear = -1;

int is\_empty() {

return front == -1 || front > rear;

}

int is\_full() {

return rear == MAX - 1;

}

void insert(int value) {

if (is\_full()) {

printf("Queue overflow! Cannot insert %d\n", value);

} else {

if (front == -1) {

front = 0;

}

queue[++rear] = value;

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

}

}

void delete() {

if (is\_empty()) {

printf("Queue underflow! Cannot delete\n");

} else {

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

front++;

if (is\_empty()) {

front = rear = -1;

}

}

}

void display() {

if (is\_empty()) {

printf("Queue is empty\n");

} else {

printf("Queue: ");

for (int i = front; i <= rear; i++) {

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

}

printf("\n");

}

}

int main() {

int choice, value;

while (1) {

printf("\nQueue Operations:\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter a value to insert: ");

scanf("%d", &value);

insert(value);

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

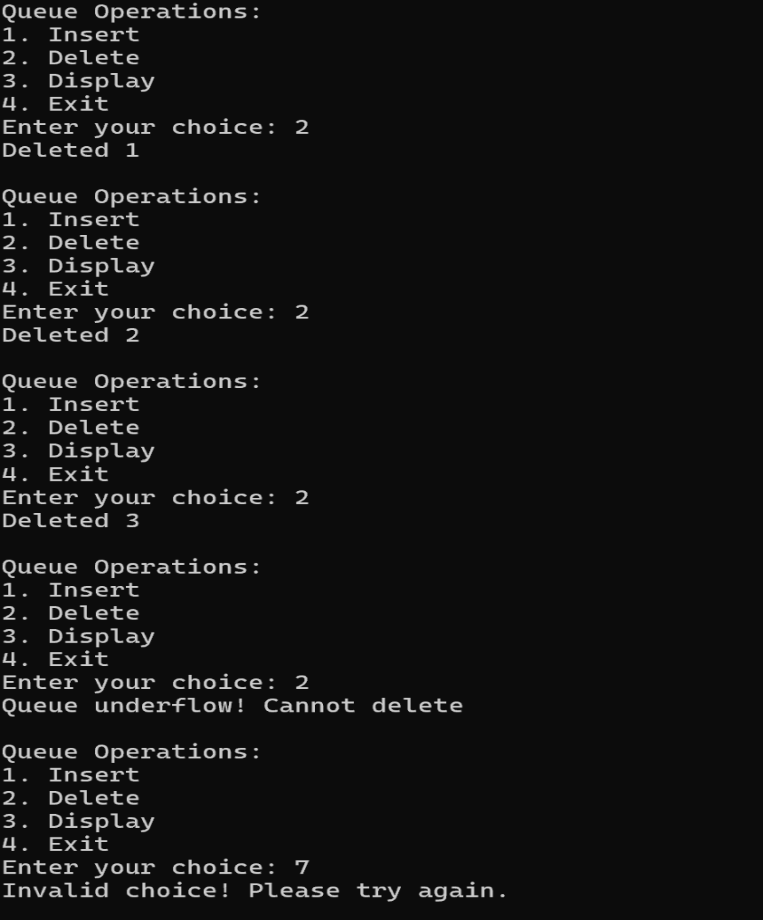
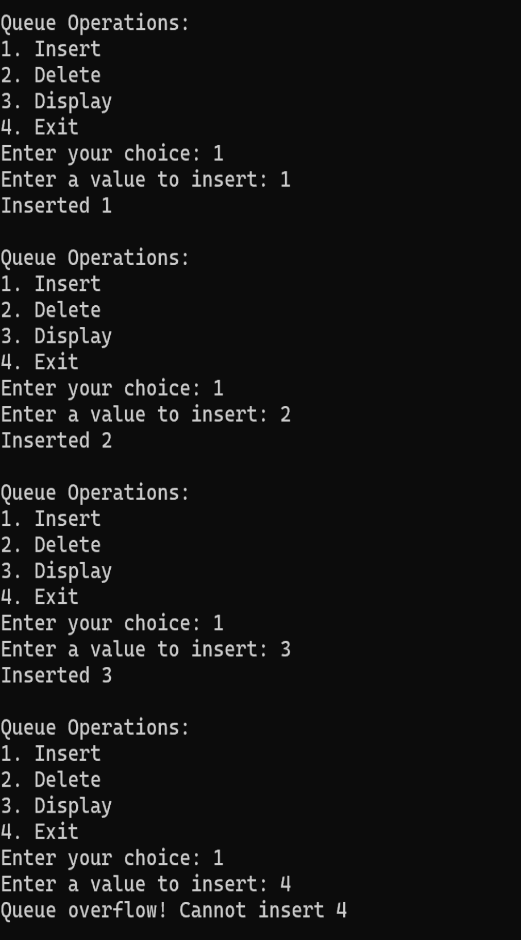
exit(0);

default:

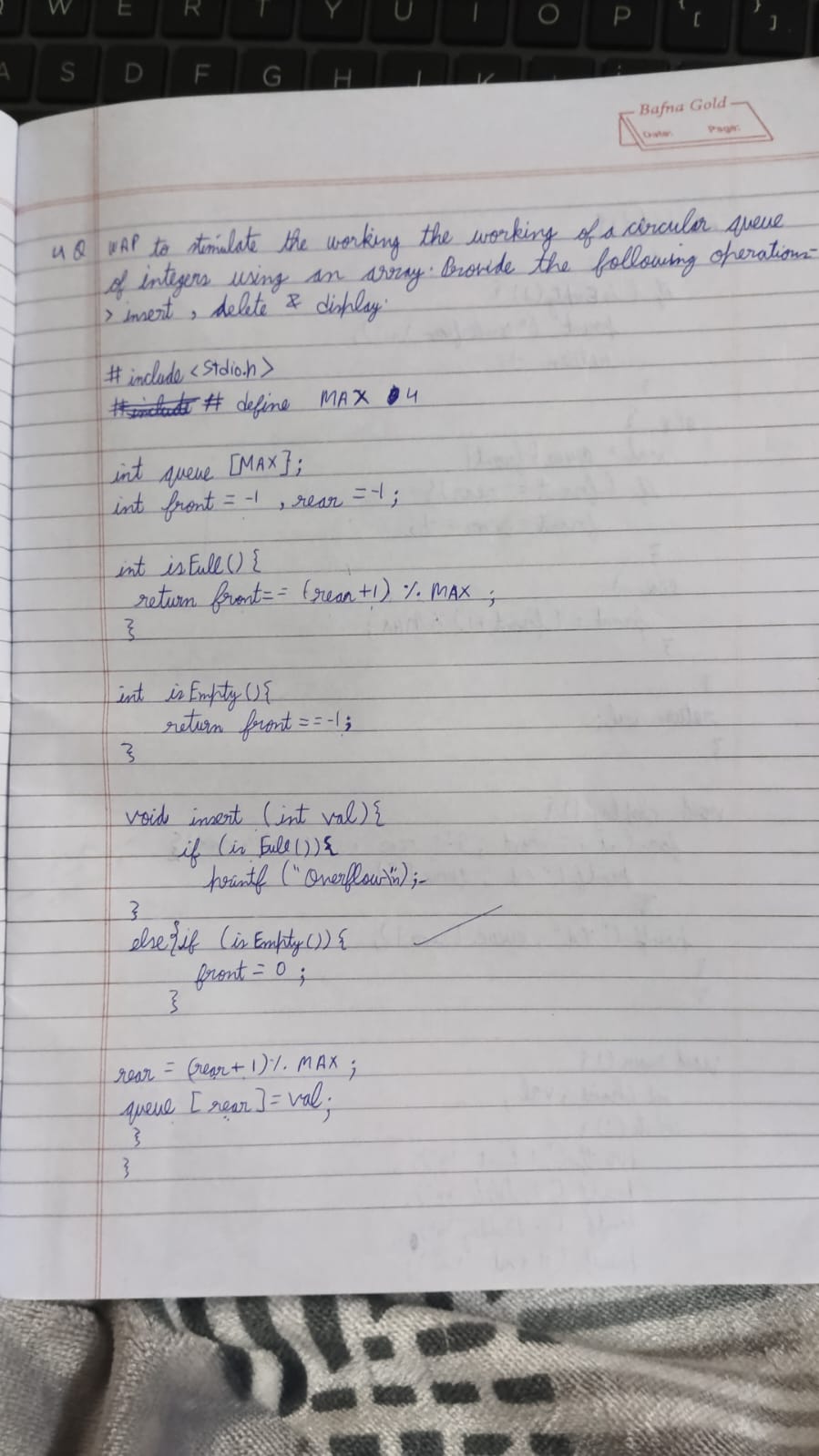
printf("Invalid choice! Please try again.\n")}

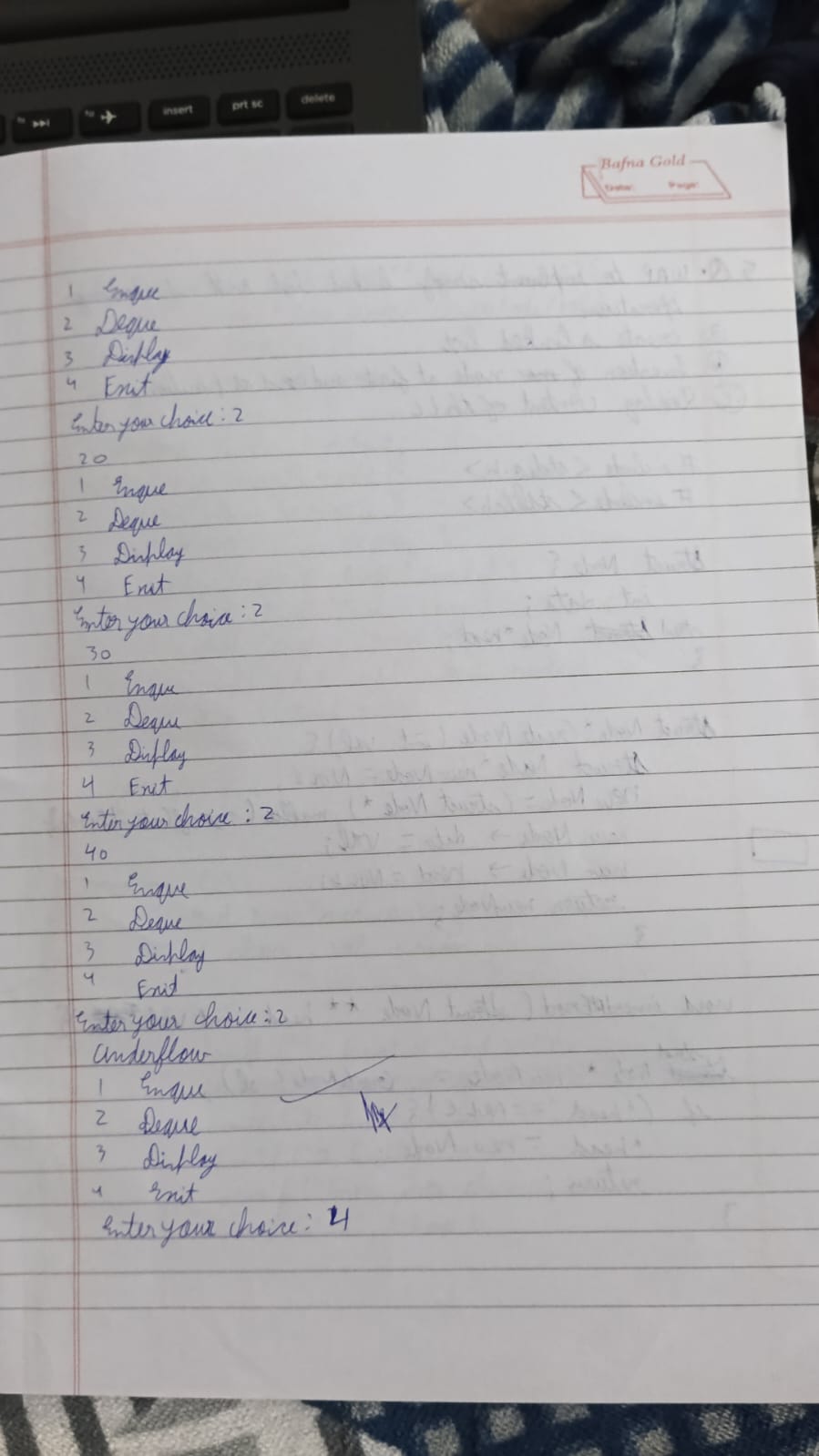
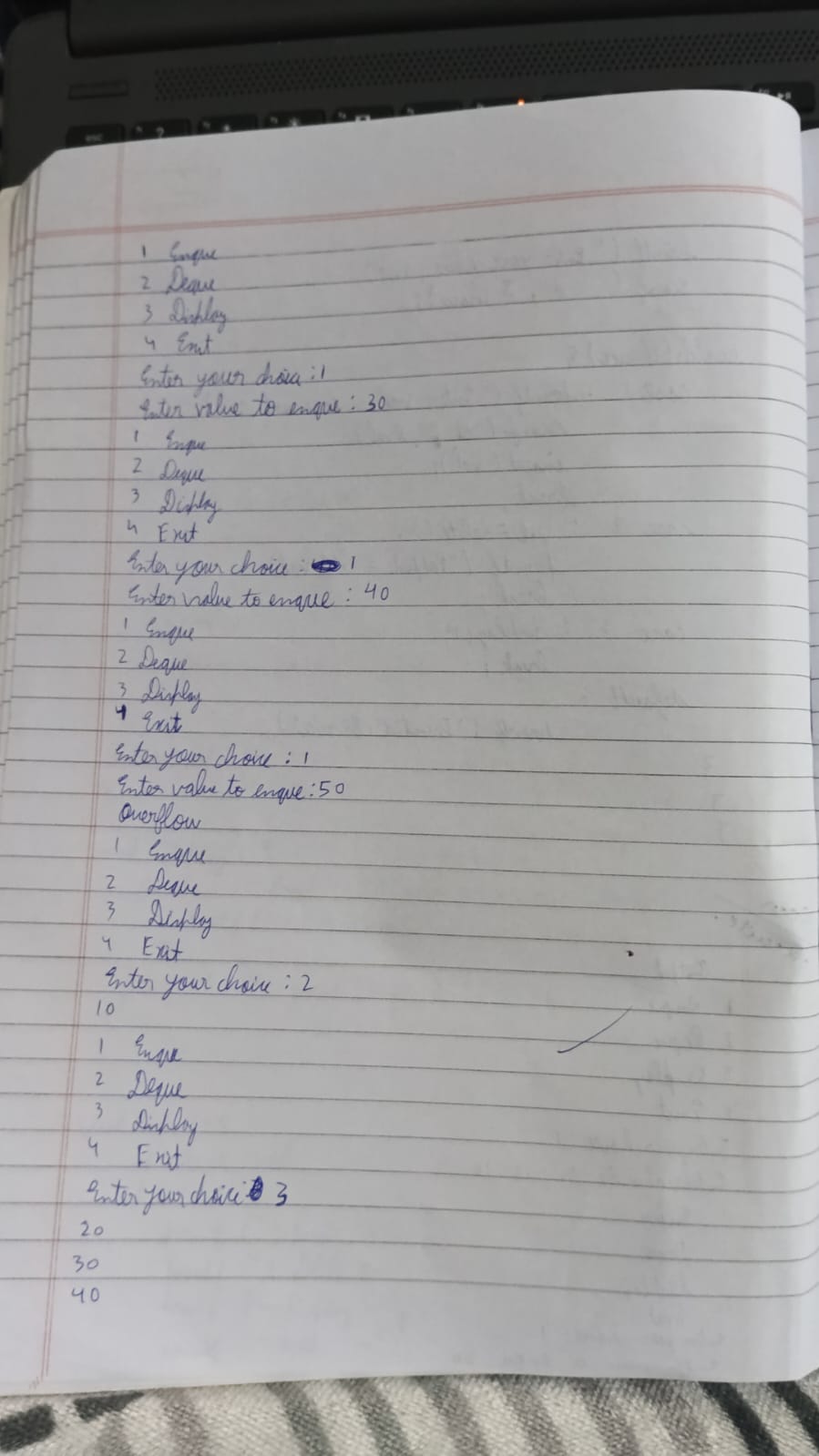
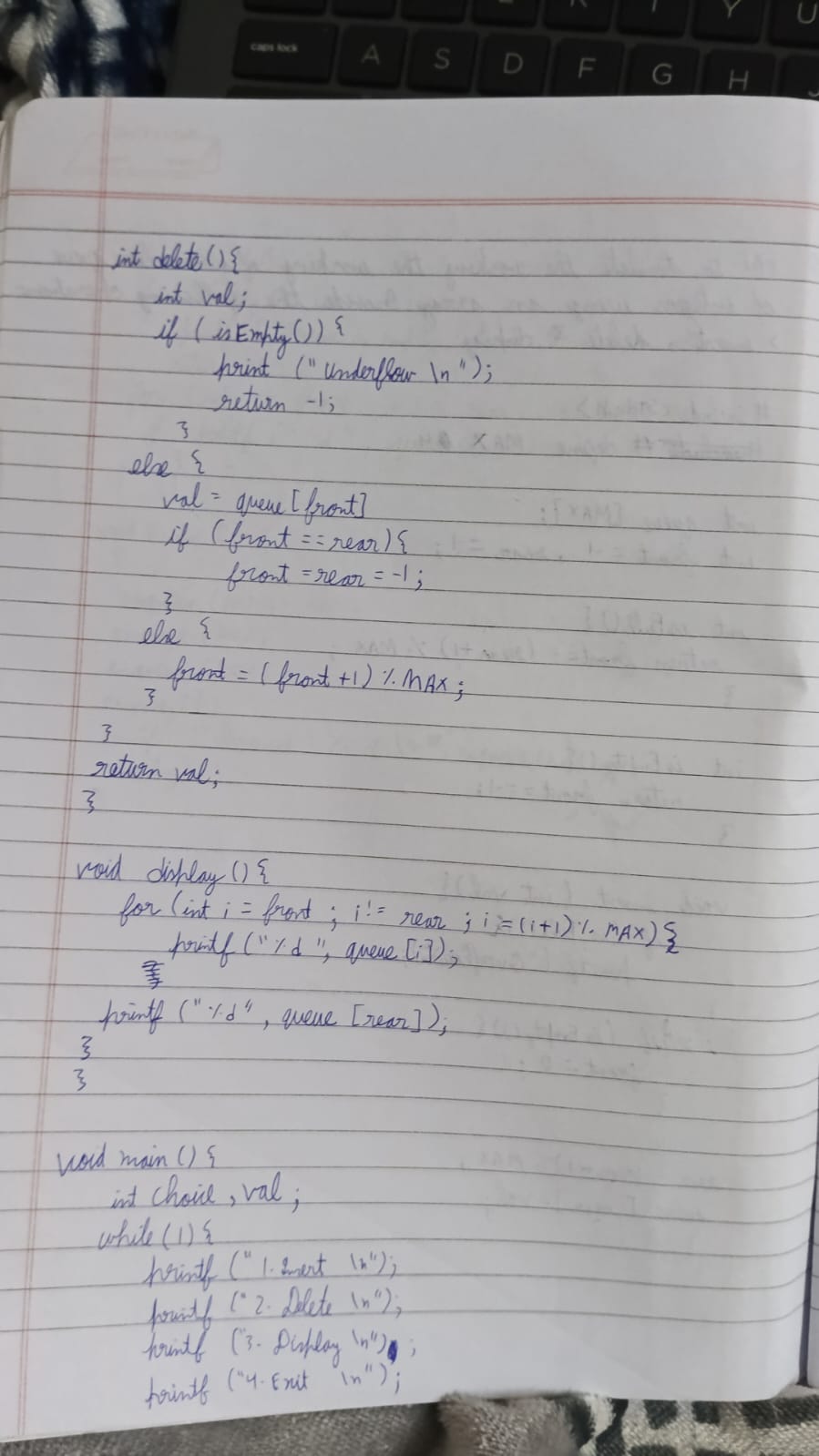
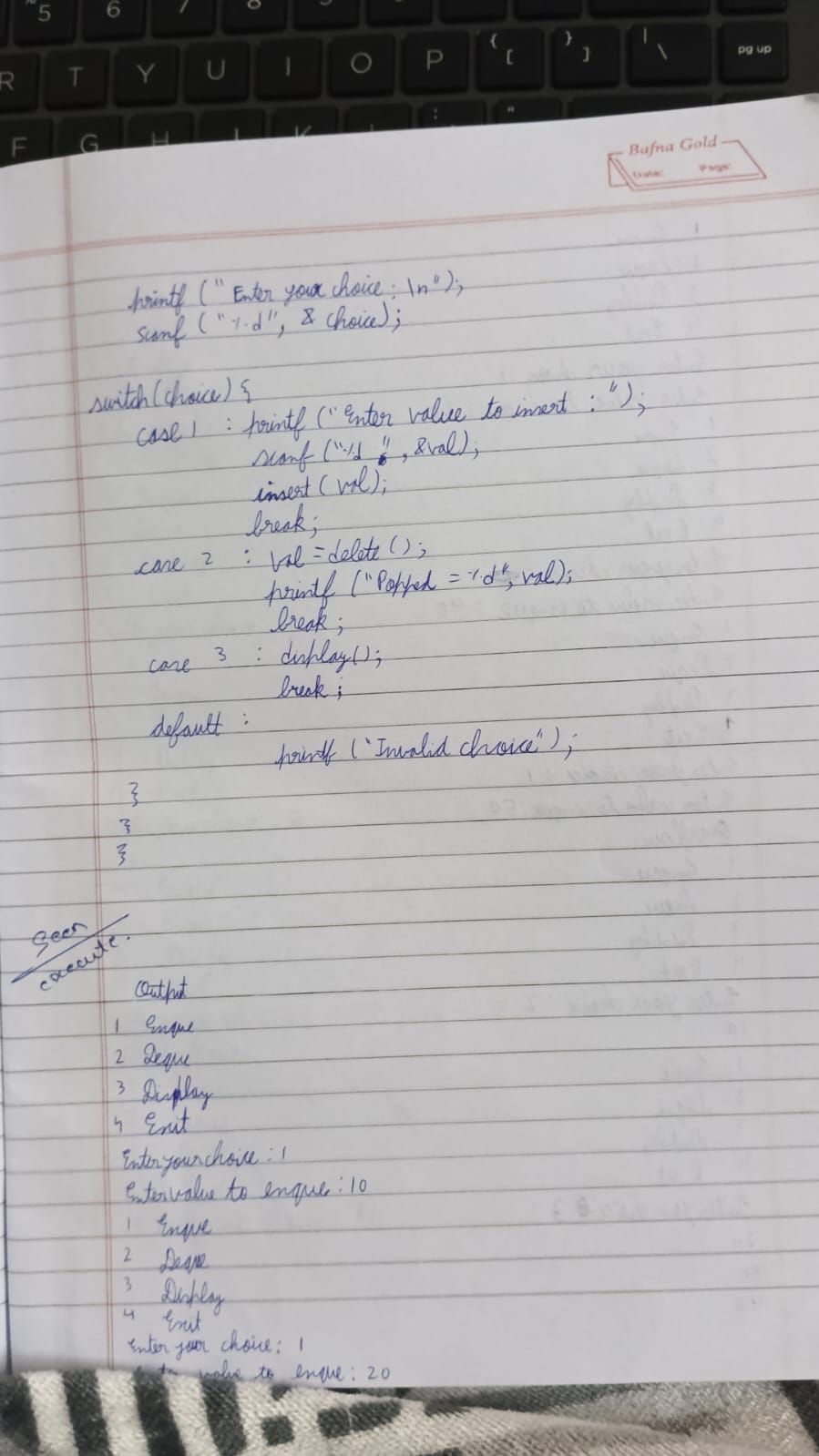
return 0;

}



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 conditions





Code:

#include <stdio.h>

#define MAX 3

int queue[MAX];

int front = -1, rear = -1;

int isfull() {

return (rear + 1) % MAX == front;

}

int isempty() {

return front == -1;

}

void insert(int val) {

if (isfull()) {

printf("Overflow\n");

return;

}

if (isempty()) {

front = 0; // Initialize front

}

rear = (rear + 1) % MAX;

queue[rear] = val;

}

int delete() {

if (isempty()) {

printf("Underflow\n");

return -1;

}

int val = queue[front];

if (front == rear) {

front = rear = -1; // Queue is now empty

} else {

front = (front + 1) % MAX;

}

return val;

}

void display() {

if (isempty()) {

printf("Queue is empty\n");

return;

}

int i = front;

while (1) {

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

if (i == rear) {

break;

}

i = (i + 1) % MAX;

}

printf("\n"); // Newline for better readability

}

int main() {

int choice, val;

while (1) {

printf("1 for insert\n2 for delete\n3 for display\n4 for exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &val);

insert(val);

break;

case 2:

val = delete();

if (val != -1) { // Check for underflow before printing

printf("Popped = %d\n", val);

}

break;

case 3:

display();

break;

case 4:

printf("Exiting...\n");

return 0;

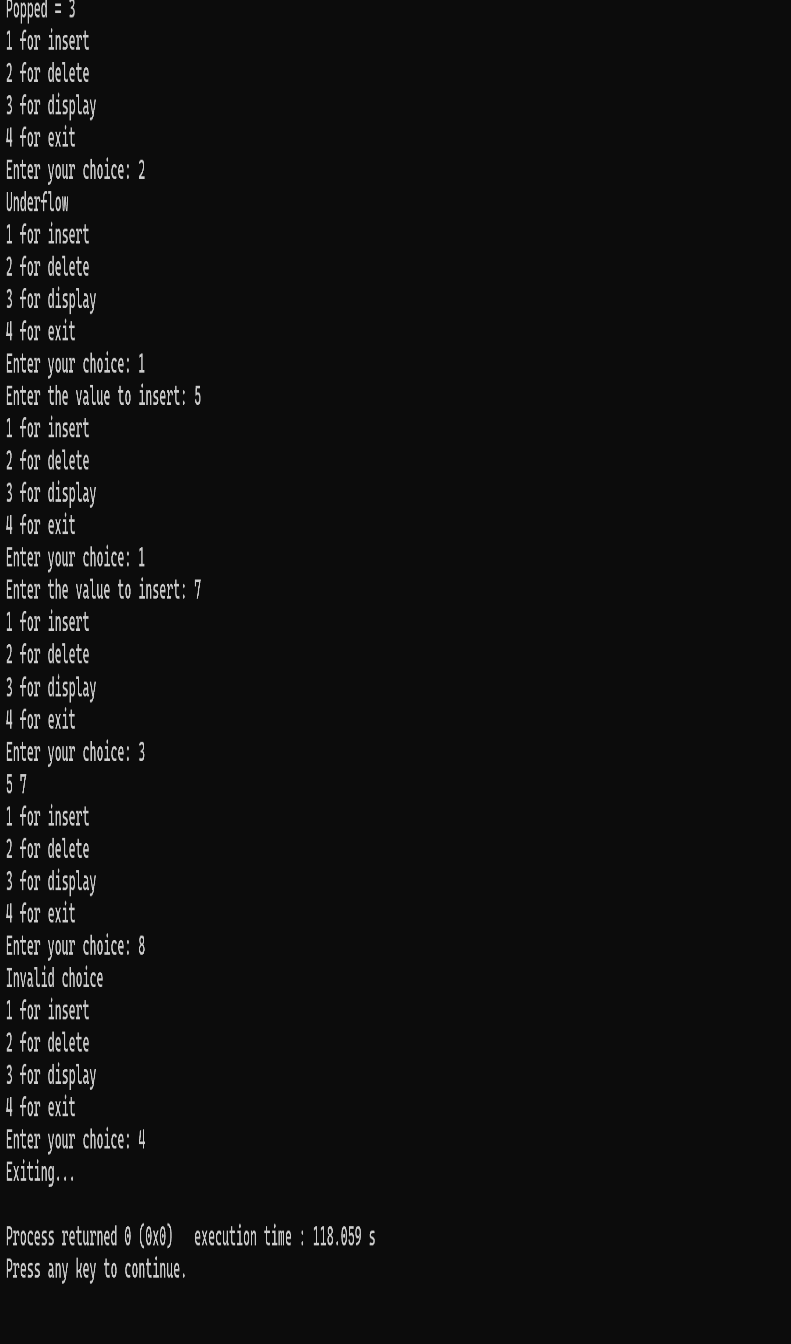
default:

printf("Invalid choice\n");

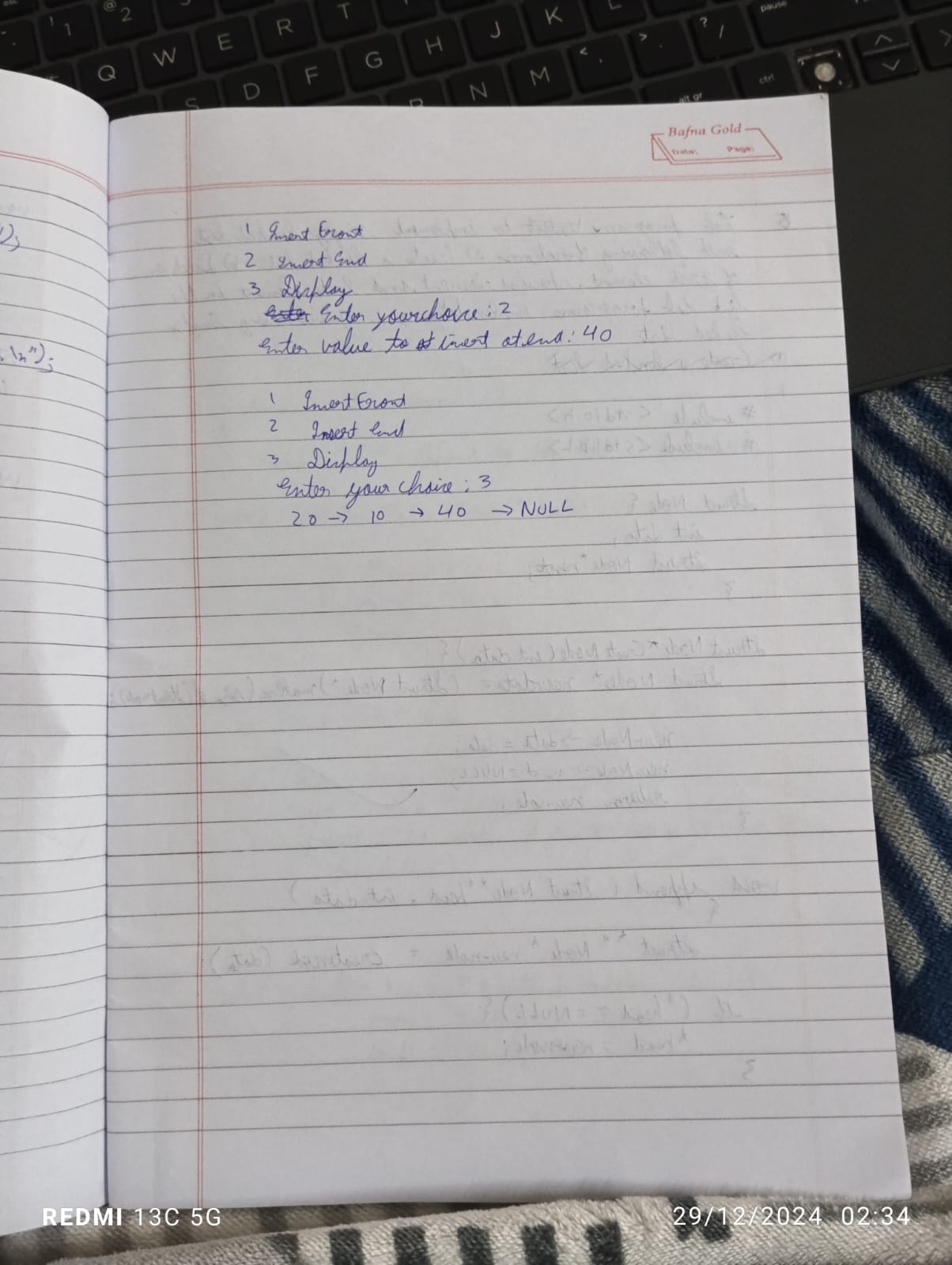
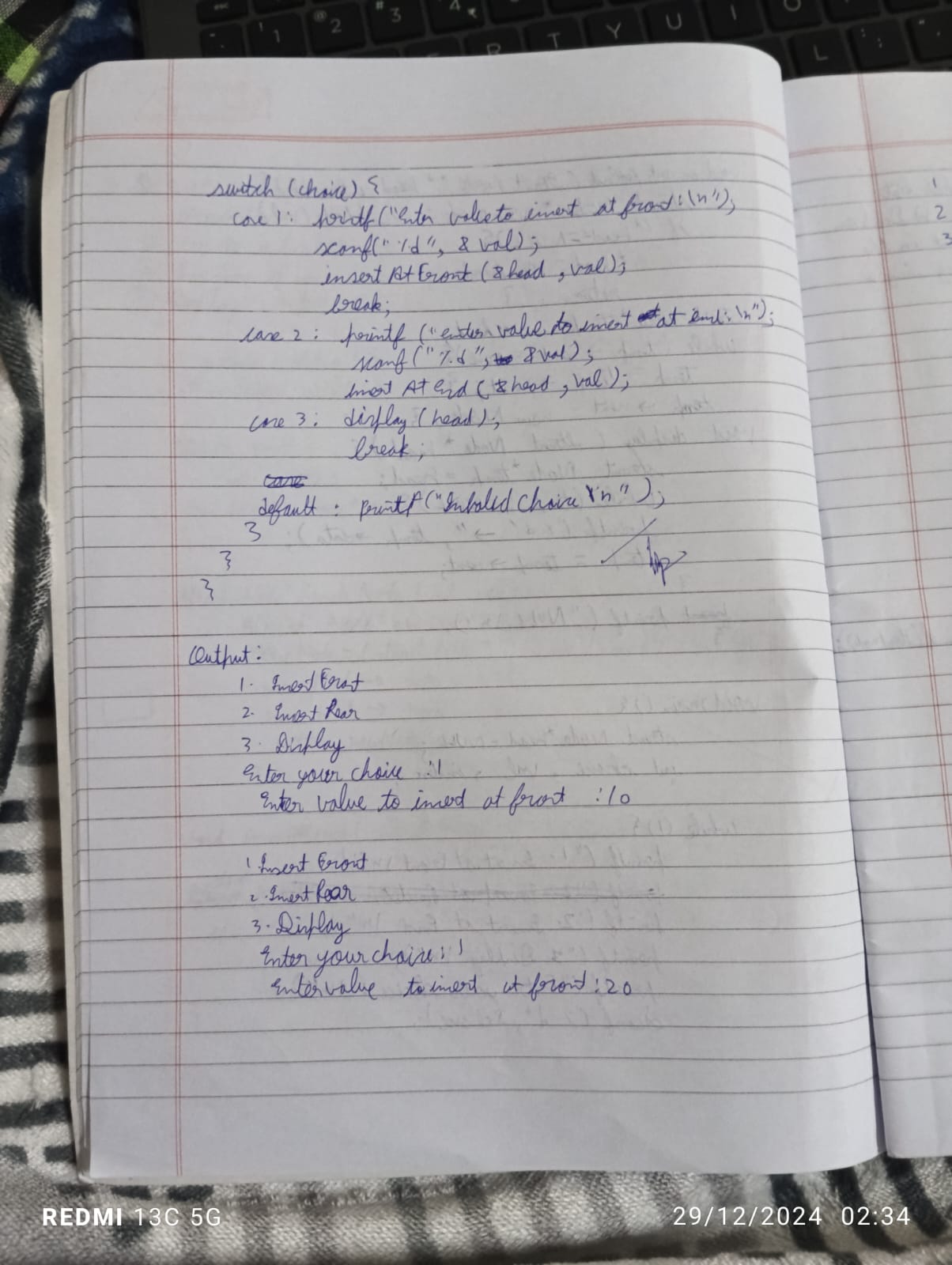
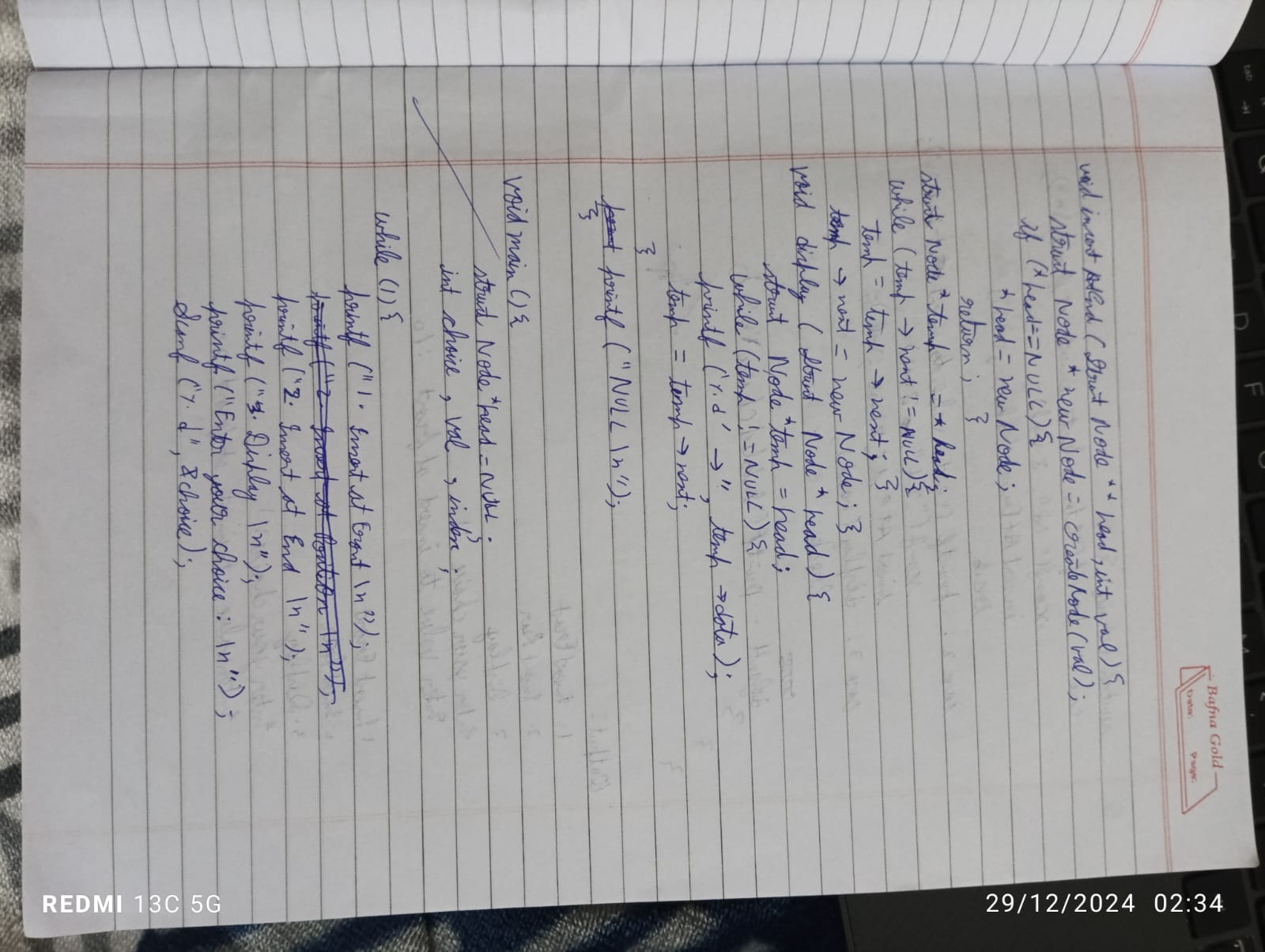
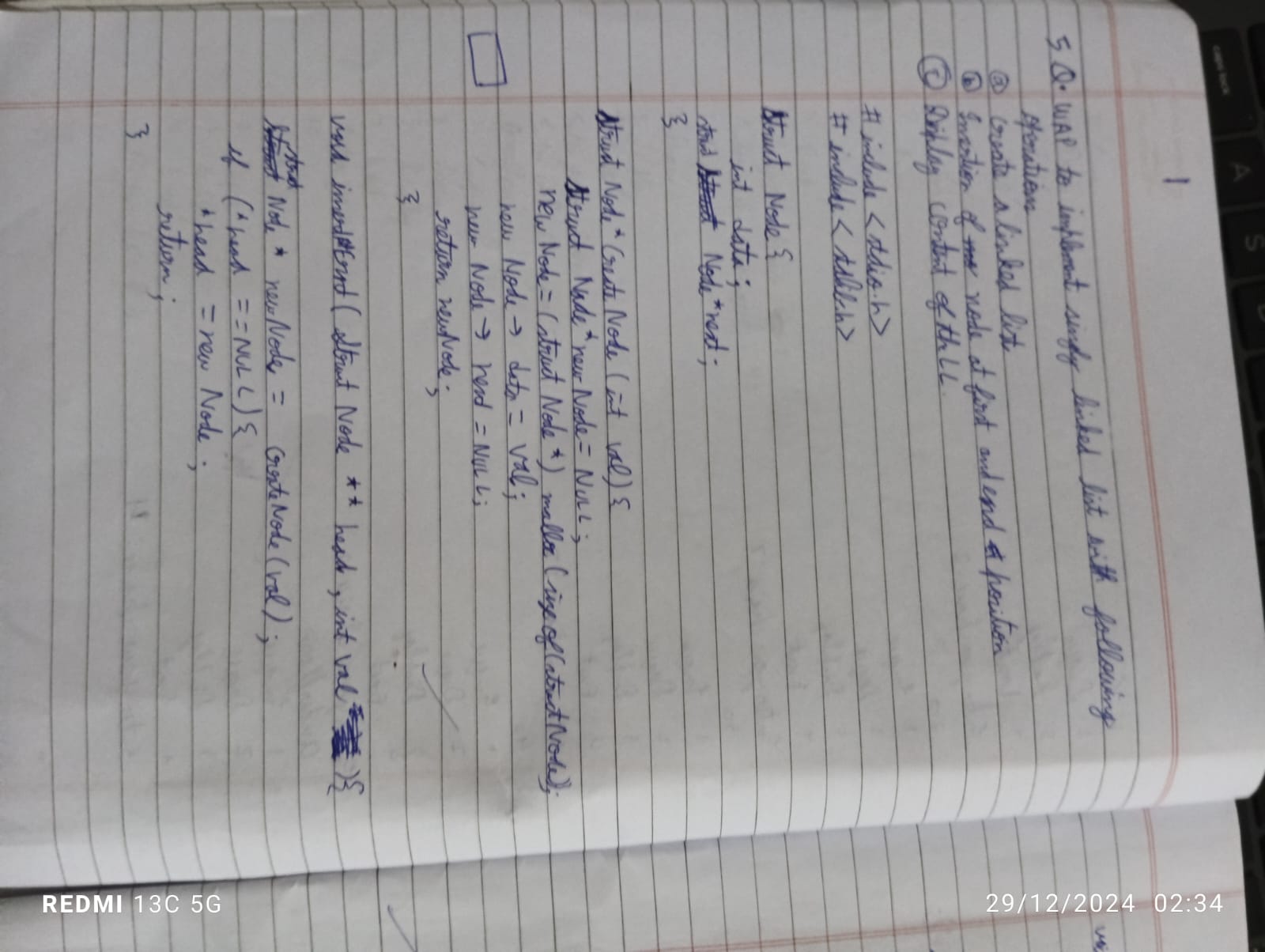
}

}

}



5)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. c) Display the contents of the linked list.



Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void create\_linked\_list(int data\_list[], int n) {

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

insert\_at\_end(data\_list[i]);

}

}

void insert\_at\_beginning(int data) {

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

new\_node->data = data;

new\_node->next = head;

head = new\_node;

}

void insert\_at\_end(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;

return;

}

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new\_node;

}

void display\_linked\_list() {

struct Node\* temp = head;

if (temp == NULL) {

printf("The linked list is empty.\n");

return;

}

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

int choice, data;

int data\_list[] = {10, 20, 30};

int n = sizeof(data\_list) / sizeof(data\_list[0]);

create\_linked\_list(data\_list, n);

while (1) {

printf("\nChoose an operation:\n");

printf("1. Display linked list\n");

printf("2. Insert at beginning\n");

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

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

display\_linked\_list();

break;

case 2:

printf("Enter a value to insert at the beginning: ");

scanf("%d", &data);

insert\_at\_beginning(data);

break;

case 3:

printf("Enter a value to insert at the end: ");

scanf("%d", &data);

insert\_at\_end(data);

break;

case 4:

exit(0);

default:

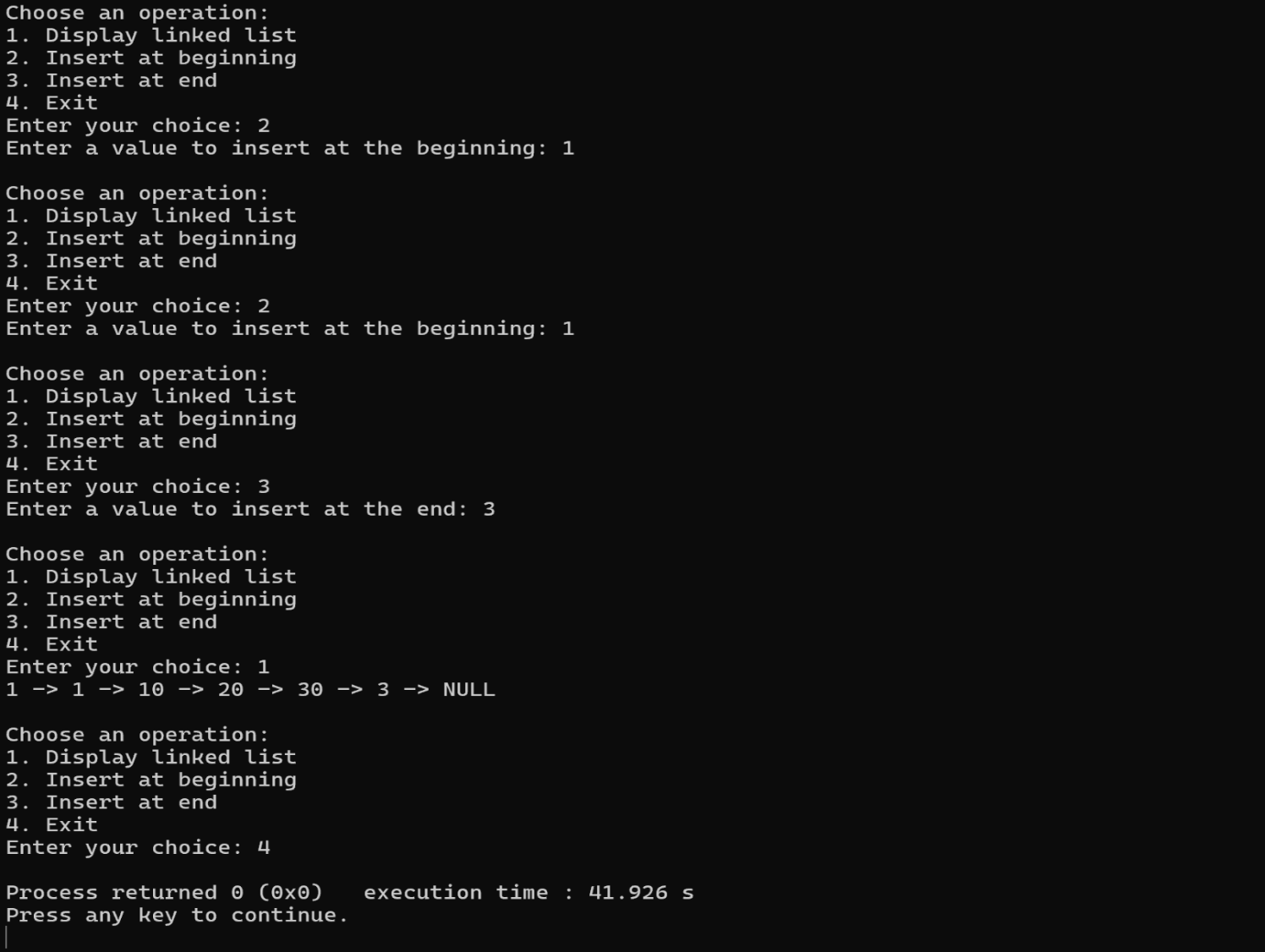
printf("Invalid choice! Please try again.\n");

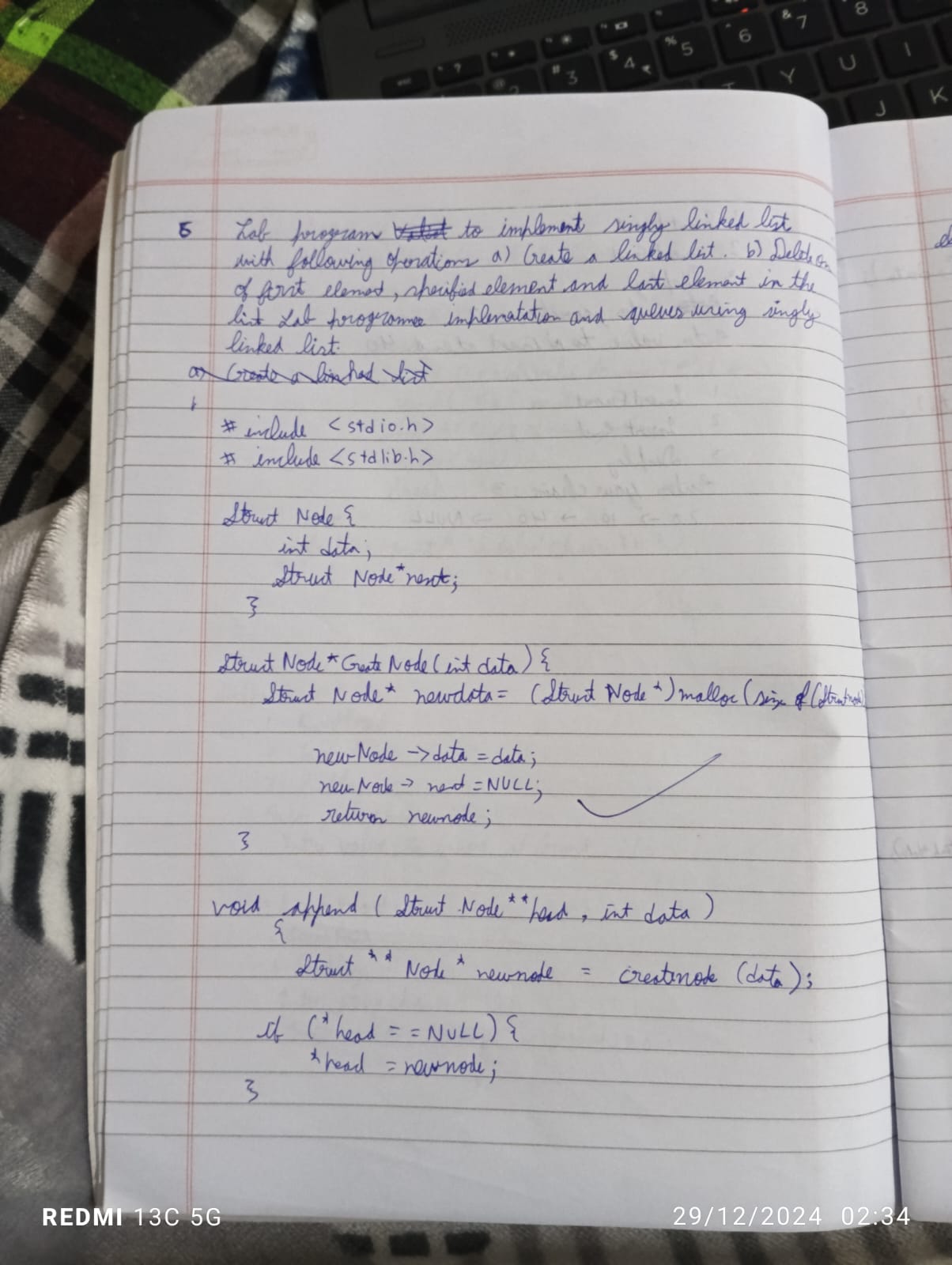
}

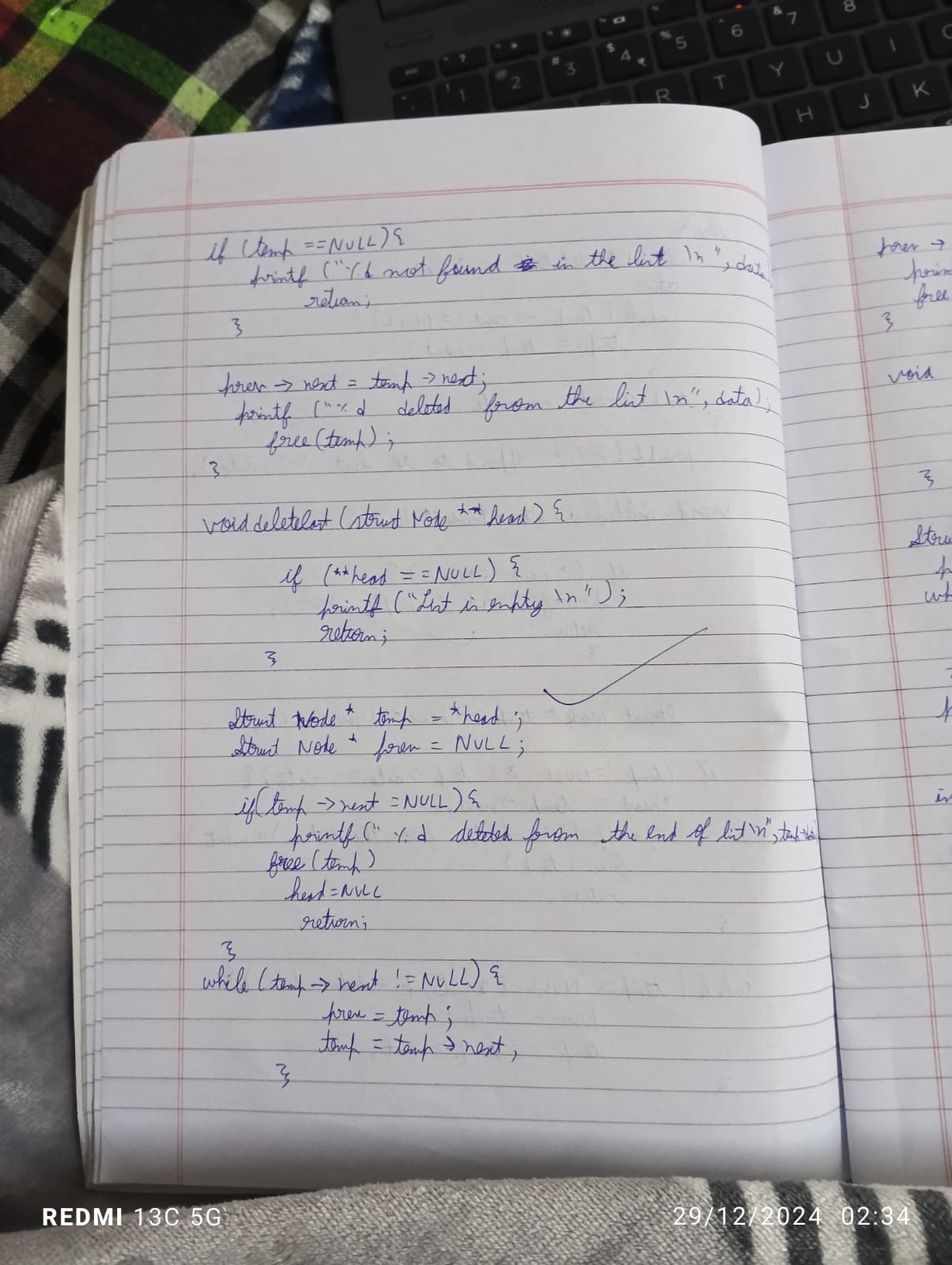
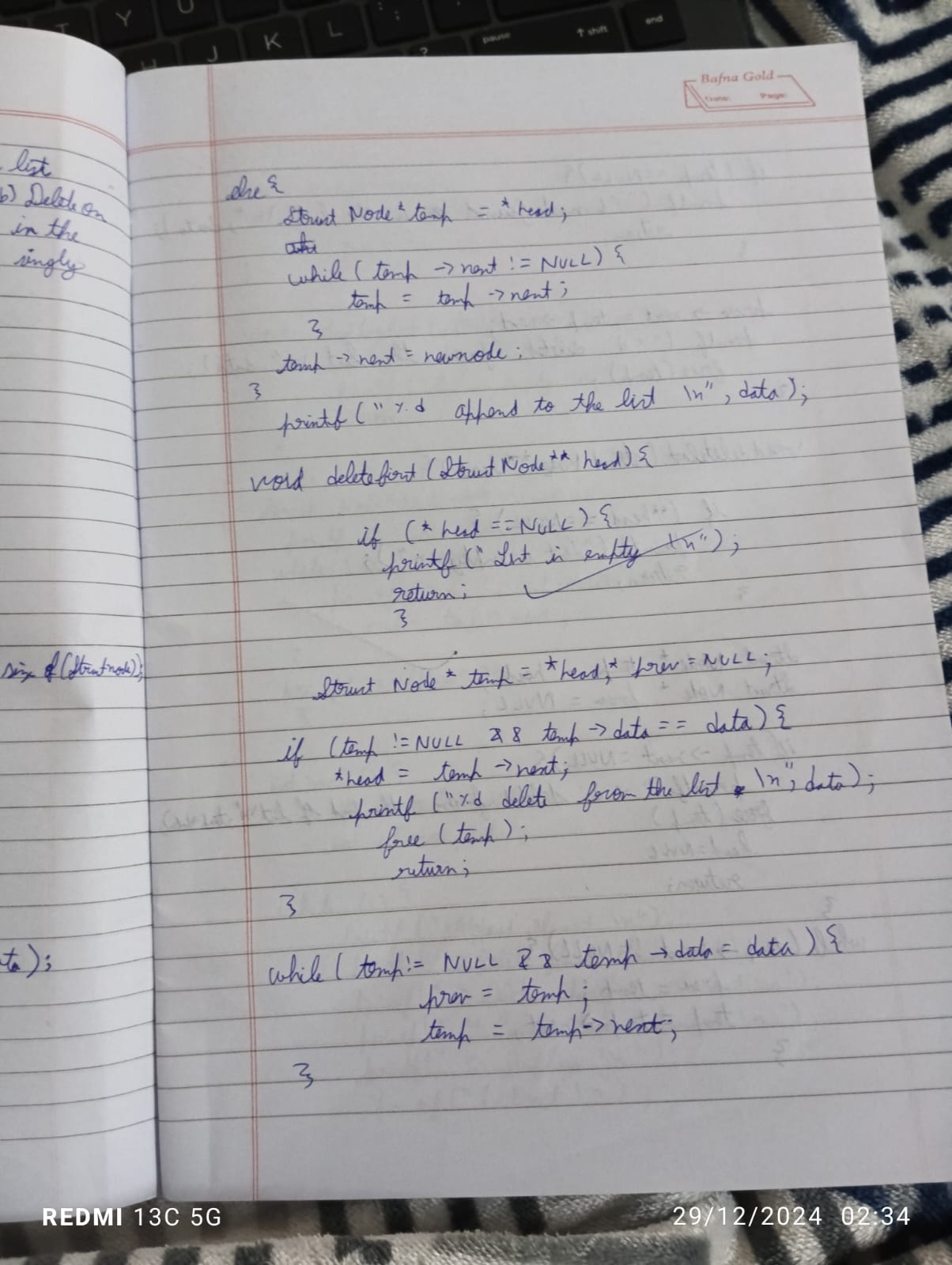
}

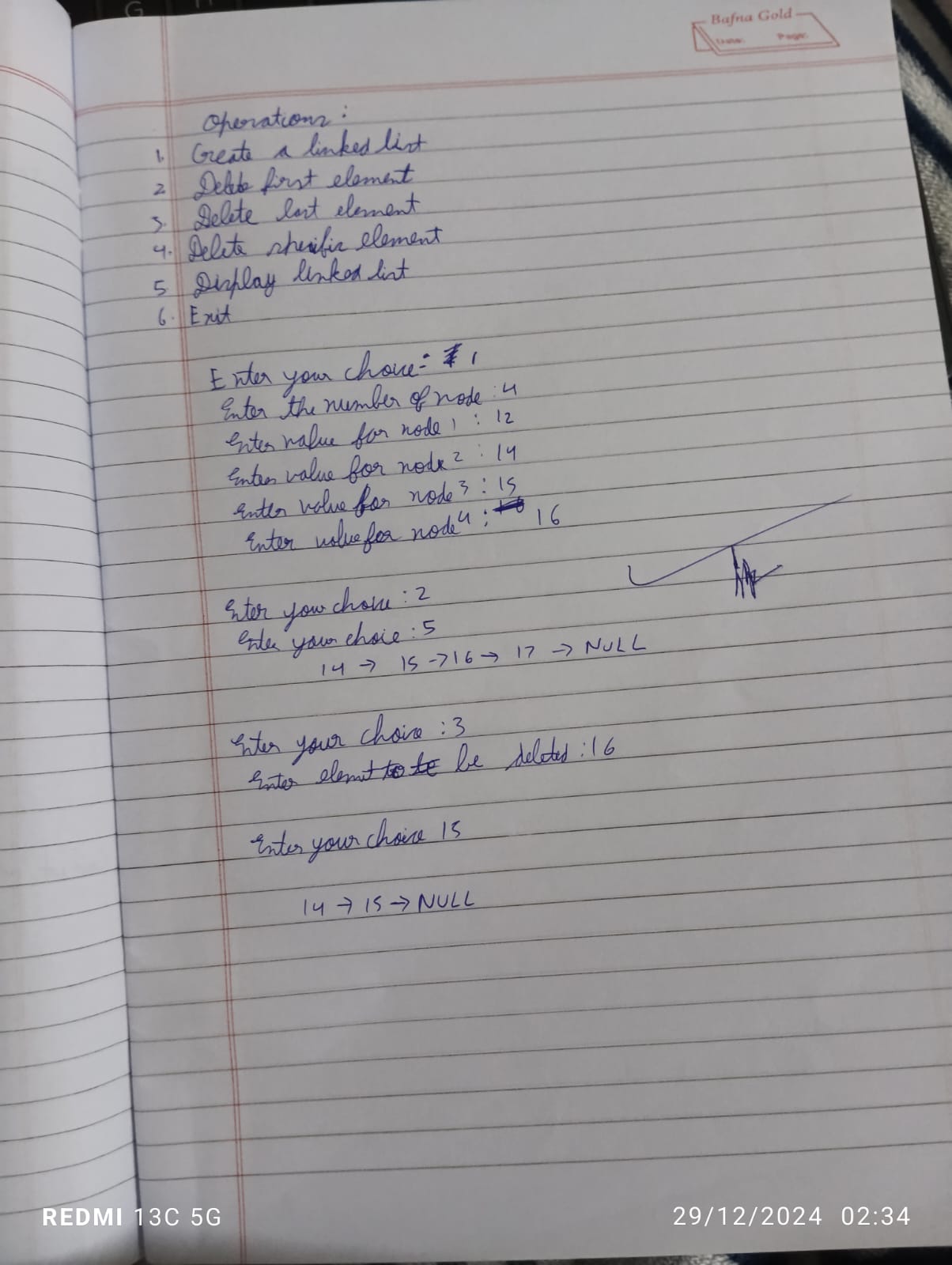
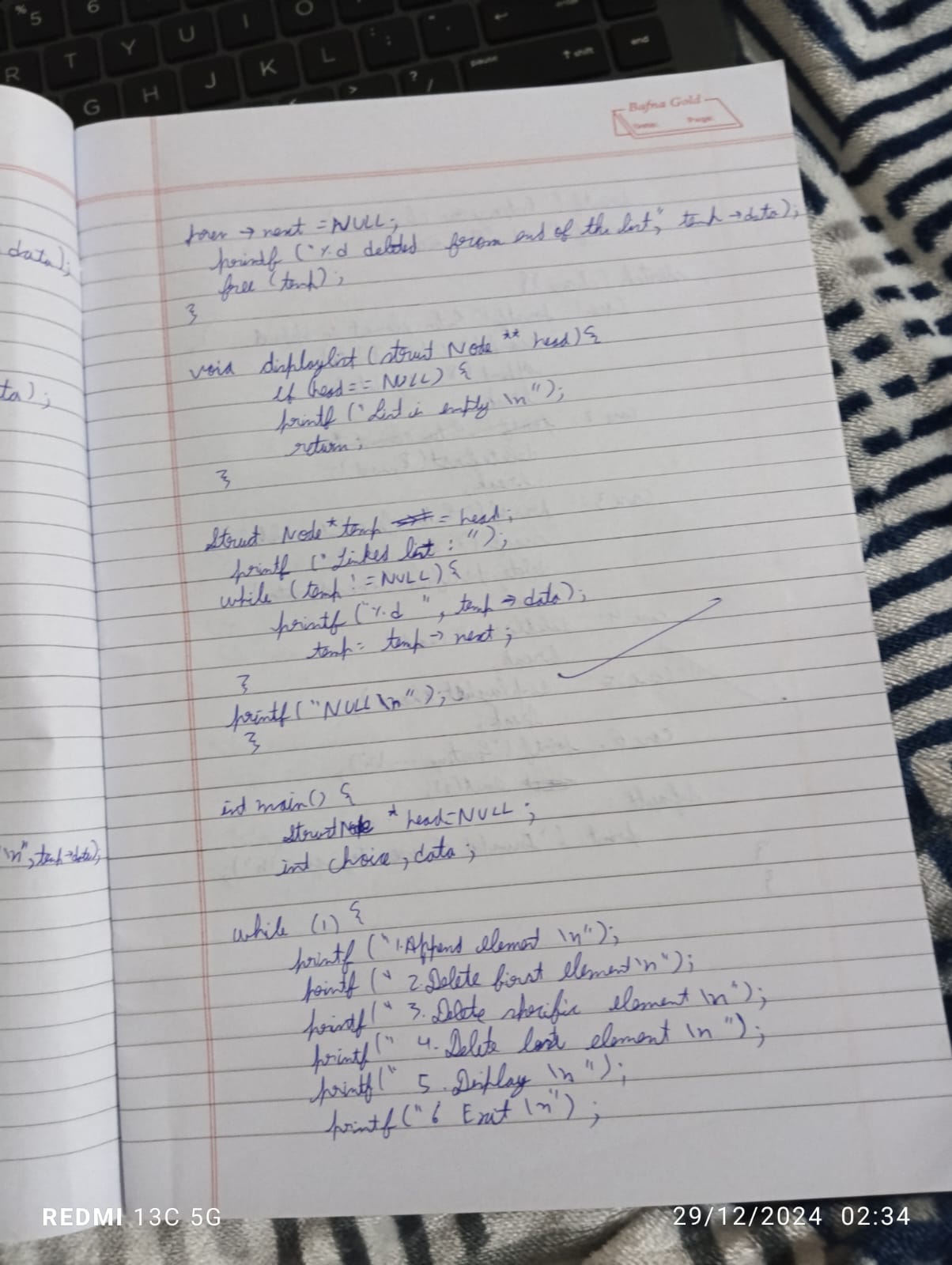
return 0;

}



6)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. 





Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

struct Node \*deleteFirst(struct Node \*head) {

if (head == NULL) return NULL;

struct Node \*ptr = head;

head = head->next;

free(ptr);

return head;

}

struct Node \*deleteByValue(struct Node \*head, int value) {

if (head == NULL) return NULL;

struct Node \*p = head;

struct Node \*q = head->next;

if (head->data == value) {

head = deleteFirst(head);

return head;

}

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

p = p->next;

q = q->next;

}

if (q != NULL && q->data == value) {

p->next = q->next;

free(q);

} else {

printf("Value not found\n");

}

return head;

}

void create(struct Node \*\*head, int data) {

struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed!\n");

return;

}

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node \*temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

struct Node \*deleteAtLast(struct Node \*head) {

if (head == NULL) return NULL;

if (head->next == NULL) {

free(head);

return NULL;

}

struct Node \*p = head;

struct Node \*q = head->next;

while (q->next != NULL) {

p = p->next;

q = q->next;

}

p->next = NULL;

free(q);

return head;

}

void linkedListDisplay(struct Node \*ptr) {

if (ptr == NULL) {

printf("The list is empty.\n");

return;

}

while (ptr != NULL) {

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

ptr = ptr->next;

}

}

int main() {

struct Node \*head = NULL;

int choice, val;

while (1) {

printf("1: Insert at end\n");

printf("2: Delete at first\n");

printf("3: Delete at last\n");

printf("4: Delete by value\n");

printf("5: Display\n");

printf("6: Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the value to insert: ");

scanf("%d", &val);

create(&head, val);

break;

case 2:

head = deleteFirst(head);

break;

case 3:

head = deleteAtLast(head);

break;

case 4:

printf("Enter the value to delete: ");

scanf("%d", &val);

head = deleteByValue(head, val);

break;

case 5:

linkedListDisplay(head);

break;

case 6:

printf("Exiting...\n");

return 0;

default:

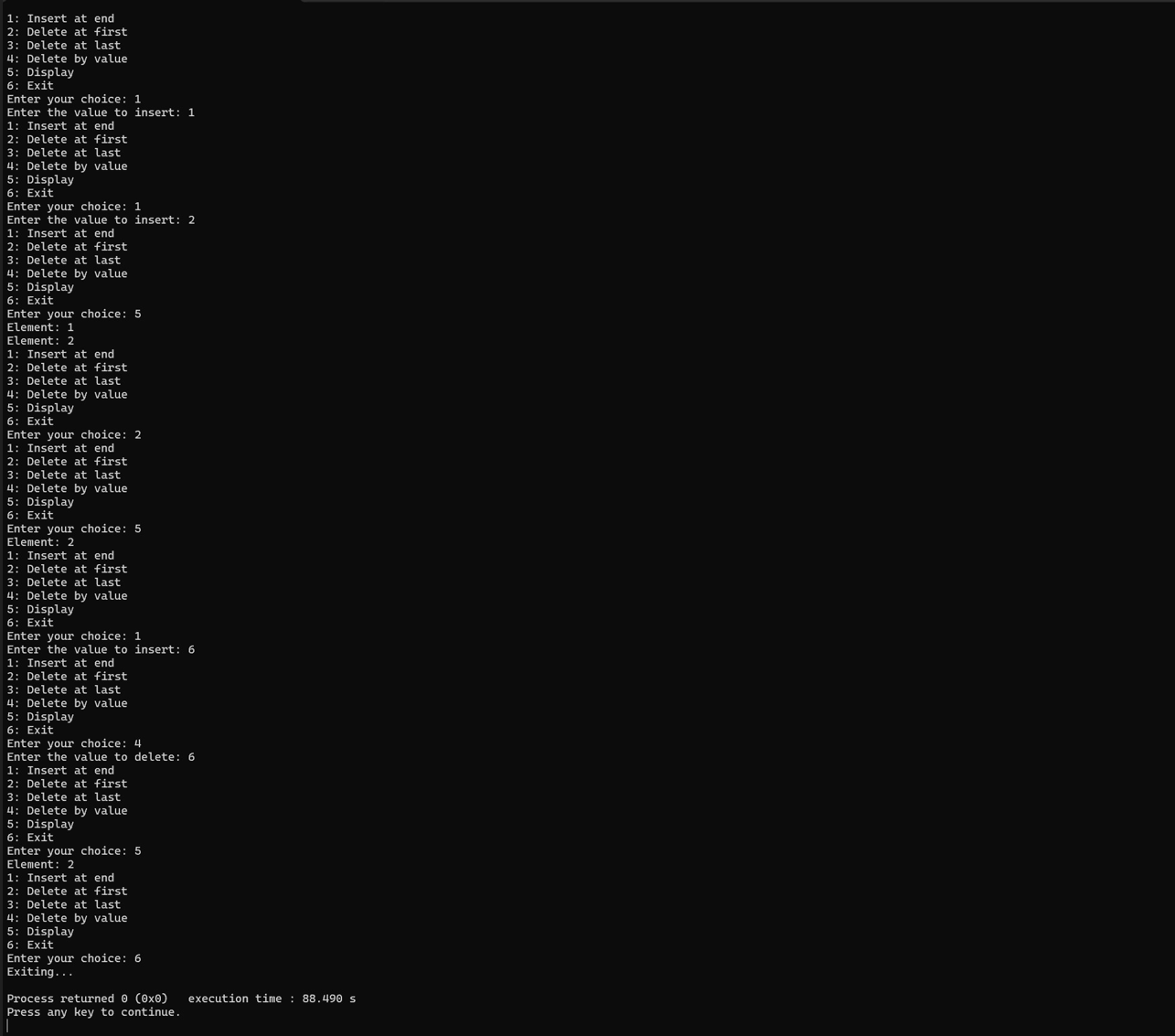
printf("Invalid choice\n");

}

}

return 0;

}



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

Code:

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\*next;

};

struct node\*head = NULL;

void append(struct node\* head , 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 sorting()

{

struct node \*ptr, \*cpt;

int temp;

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

return;

}

for (ptr = head; ptr != NULL; ptr = ptr->next) {

for (cpt = ptr->next; cpt != NULL; cpt = cpt->next) {

if (ptr->data > cpt->data) {

temp = ptr->data;

ptr->data = cpt->data;

cpt->data = temp;

}

}

}

}

void concat(struct node \*head1 , struct node \*head2)

{

struct node \* ptr = NULL;

ptr = head1;

while(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr->next = head2;

}

void reverse(struct node\*\* head) {

struct node\* prev = NULL;

struct node\* curr = \*head;

struct node\* next = NULL;

while (curr != NULL) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

\*head = prev;

}

void display()

{

struct node \* temp;

temp = head ;

if (temp != NULL)

{

printf("list is empty");

}

else {

while (temp != NULL)

{

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

}

}

}

int main() {

struct node \*list1 = NULL;

struct node \*list2 = NULL;

int choice, data;

while (1) {

printf("1 for append to list1\n");

printf("2 for append to list2\n");

printf("3 for concatenation\n");

printf("4 for display list1\n");

printf("5 for display list2\n");

printf("6 for reverse list1\n");

printf("7 for reverse list2\n");

printf("8 for exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Add to list1: ");

scanf("%d", &data);

append(&list1, data);

break;

case 2:

printf("Add to list2: ");

scanf("%d", &data);

append(&list2, data);

break;

case 3:

concat(&list1, list2);

break;

case 4:

printf("List1: ");

display(list1);

break;

case 5:

printf("List2: ");

display(list2);

break;

case 6:

printf("Reversing list1...\n");

reverse(&list1);

break;

case 7:

printf("Reversing list2...\n");

reverse(&list2);

break;

case 8:

printf("Exiting...\n");

return 0;

default:

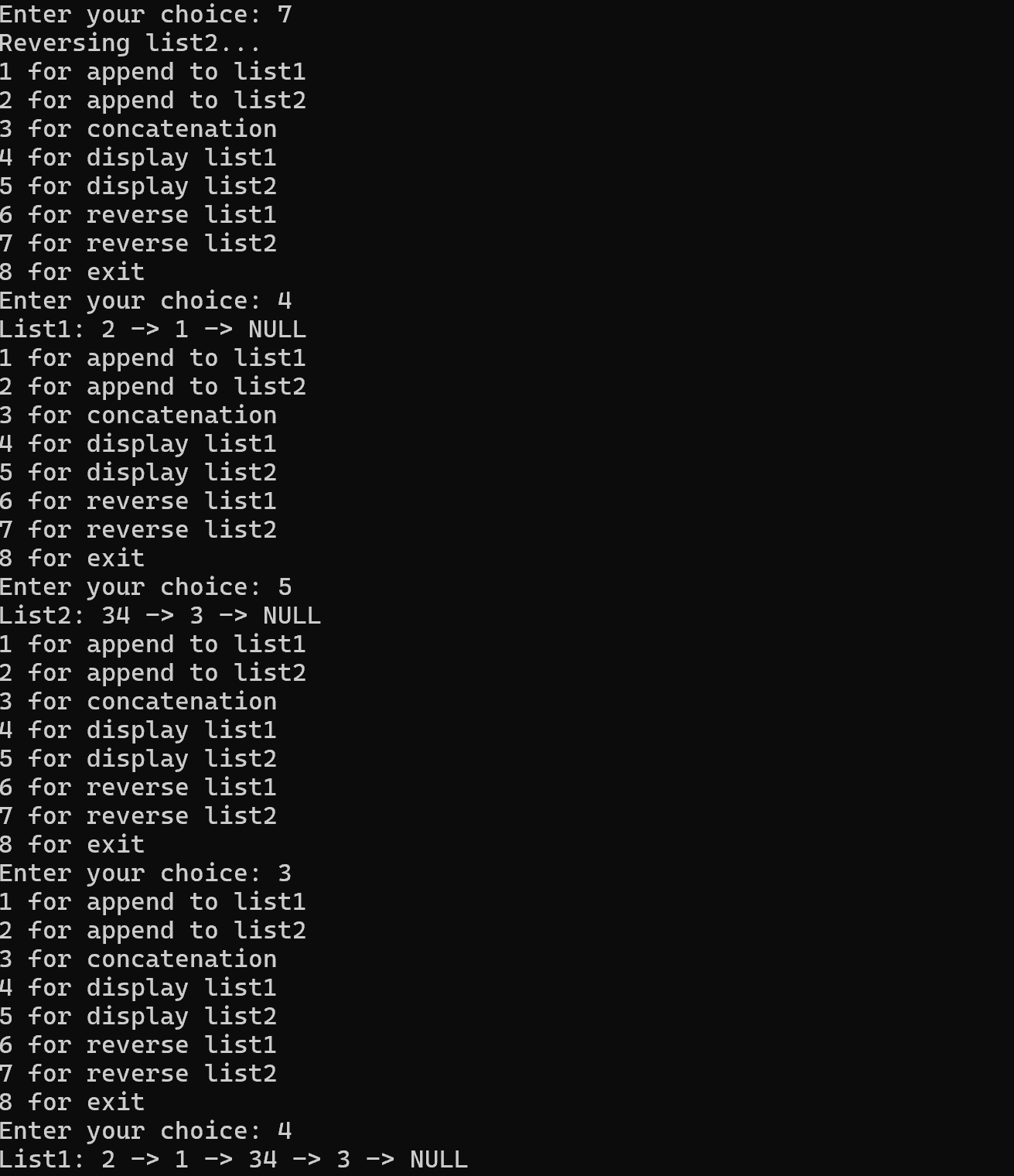
printf("Invalid choice\n");

break;

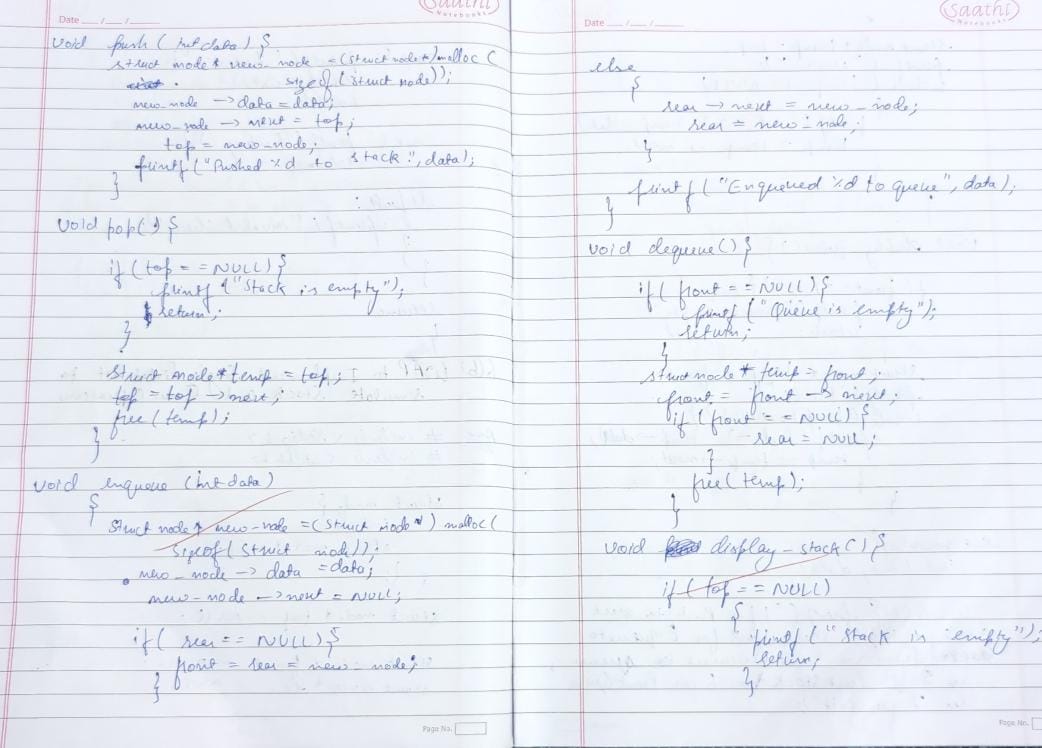
}

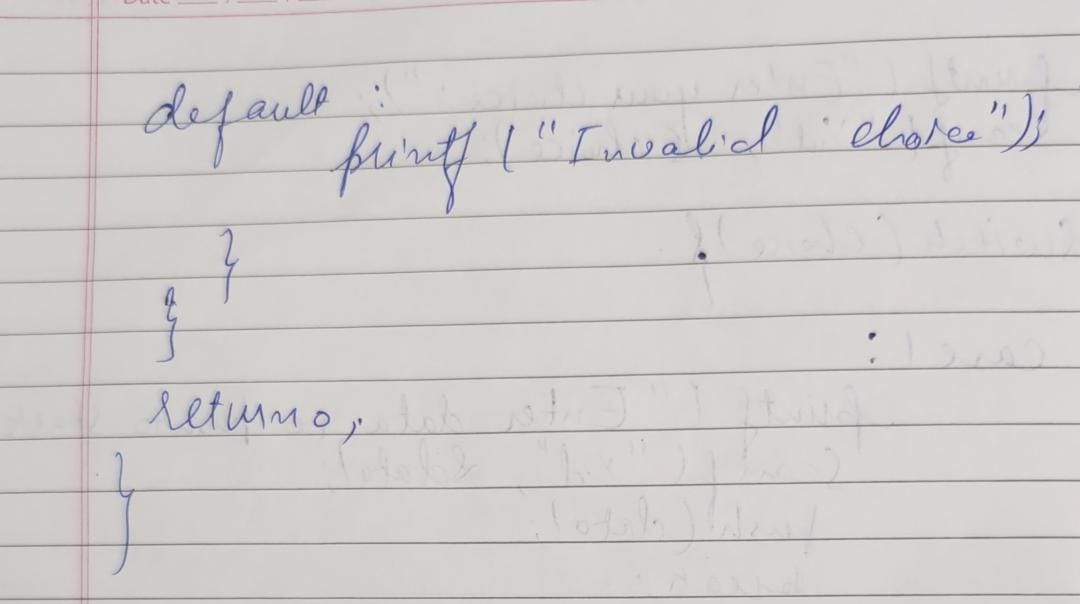
}

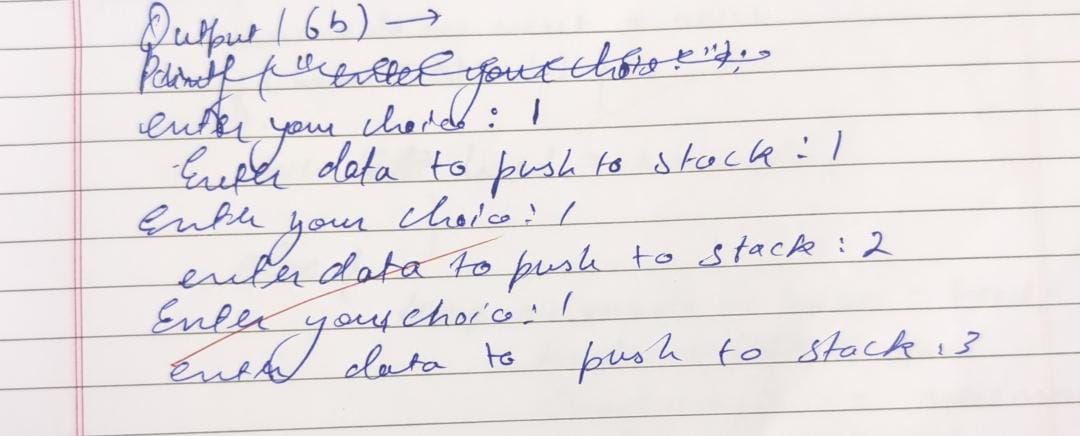
}

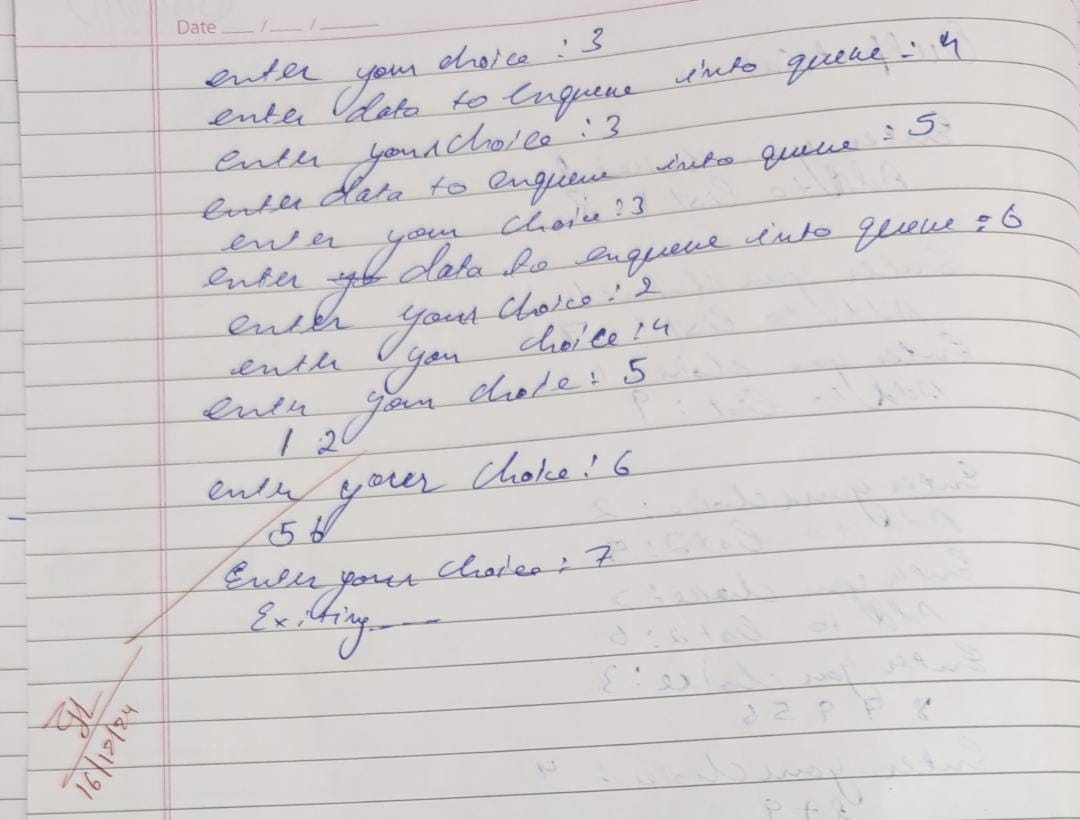


6) b) WAP to Implement Single Link List to simulate Stack & Queue Operations.









Code:

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* next;

};

struct node\* top = NULL;

struct node\* front = NULL;

struct node\* rear = NULL;

void push(int data) {

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

new\_node->data = data;

new\_node->next = top;

top = new\_node;

printf("Pushed %d to stack\n", data);

}

void pop() {

if (top == NULL) {

printf("Stack is empty\n");

return;

}

struct node\* temp = top;

top = top->next;

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

free(temp);

}

void display\_stack() {

if (top == NULL) {

printf("Stack is empty\n");

return;

}

struct node\* temp = top;

printf("Stack: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void enqueue(int data) {

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

new\_node->data = data;

new\_node->next = NULL;

if (rear == NULL) {

front = rear = new\_node;

} else {

rear->next = new\_node;

rear = new\_node;

}

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

}

void dequeue() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

struct node\* temp = front;

front = front->next;

if (front == NULL) {

rear = NULL;

}

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

free(temp);

}

void display\_queue() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

struct node\* temp = front;

printf("Queue: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

int choice, data;

while (1) {

printf("\n1. 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: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to push to stack: ");

scanf("%d", &data);

push(data);

break;

case 2:

pop();

break;

case 3:

display\_stack();

break;

case 4:

printf("Enter data to enqueue to queue: ");

scanf("%d", &data);

enqueue(data);

break;

case 5:

dequeue();

break;

case 6:

display\_queue();

break;

case 7:

printf("Exiting...\n");

return 0;

default:

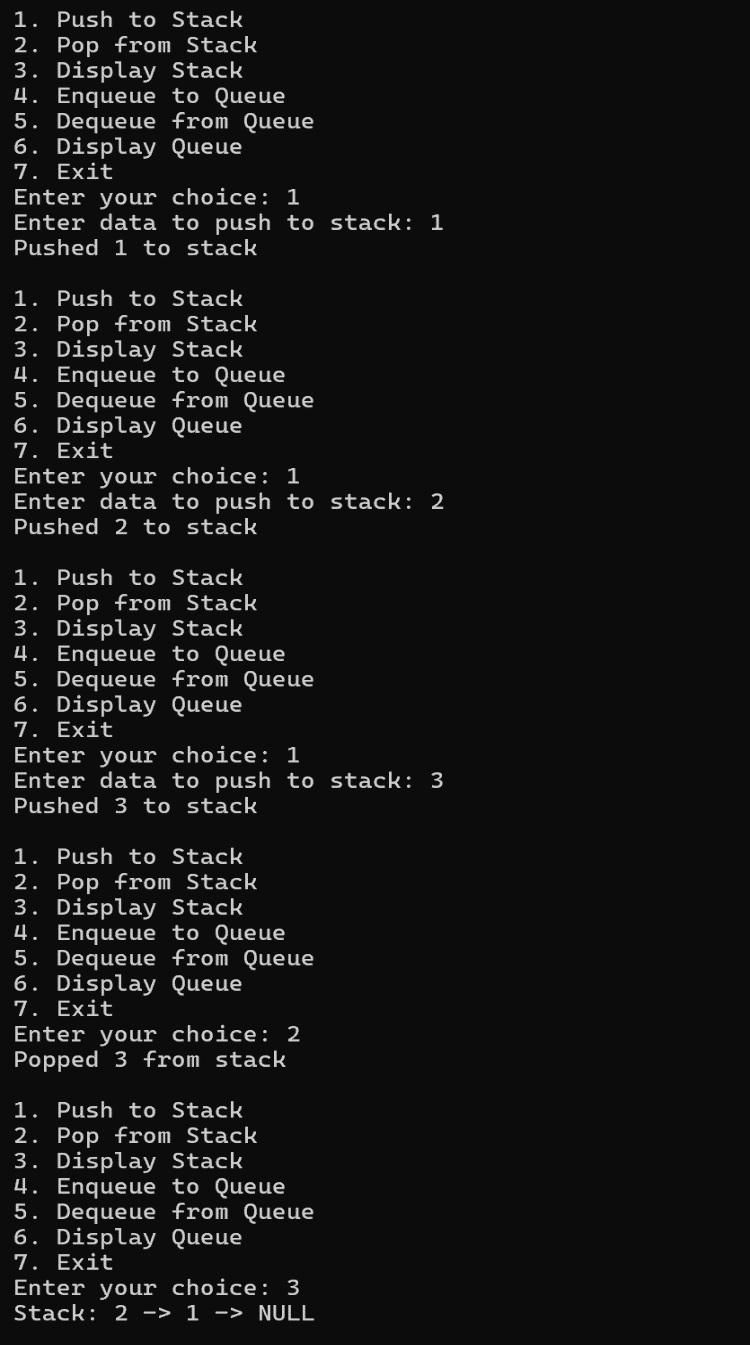
printf("Invalid choice! Please try again.\n");

}

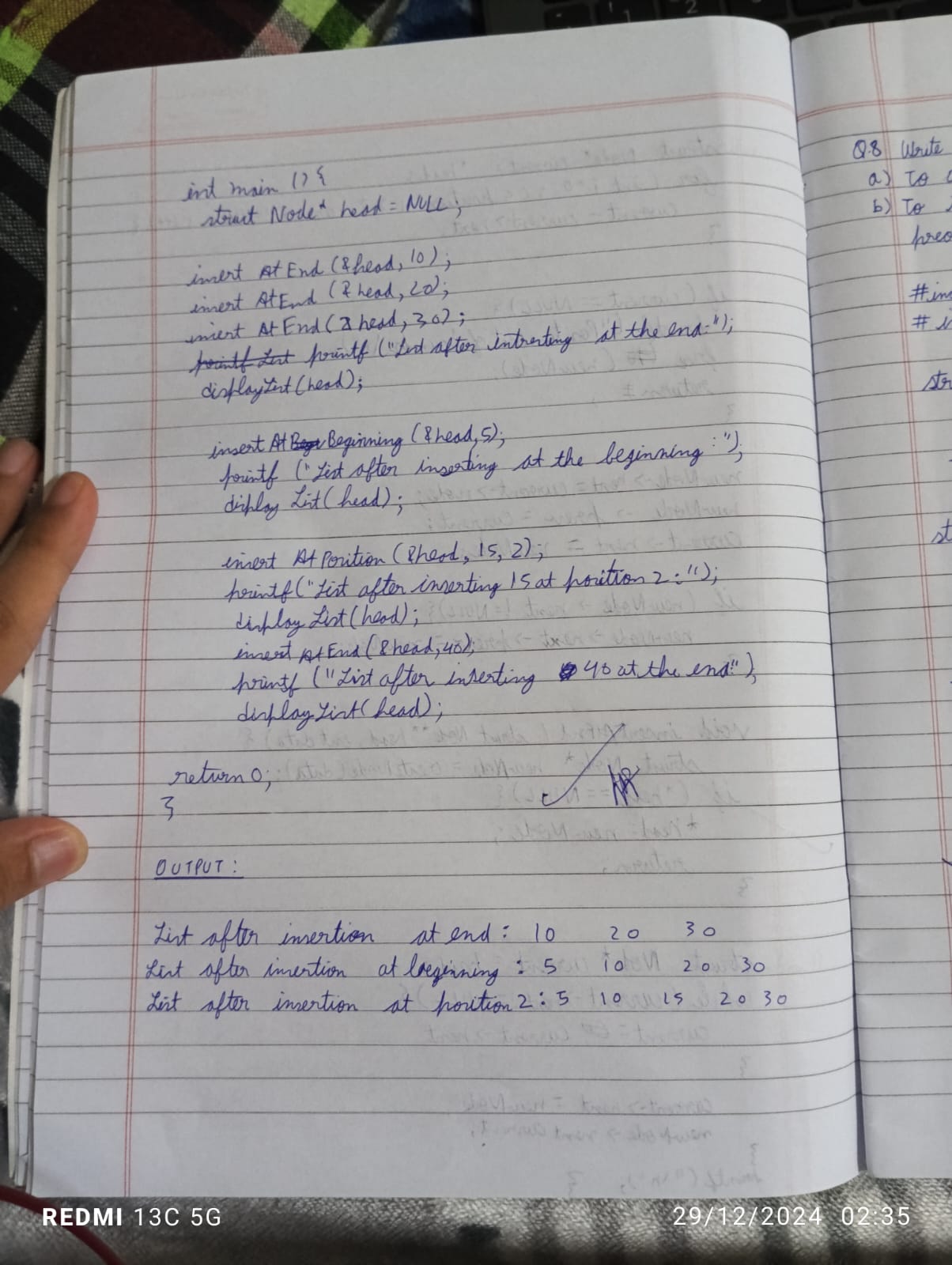
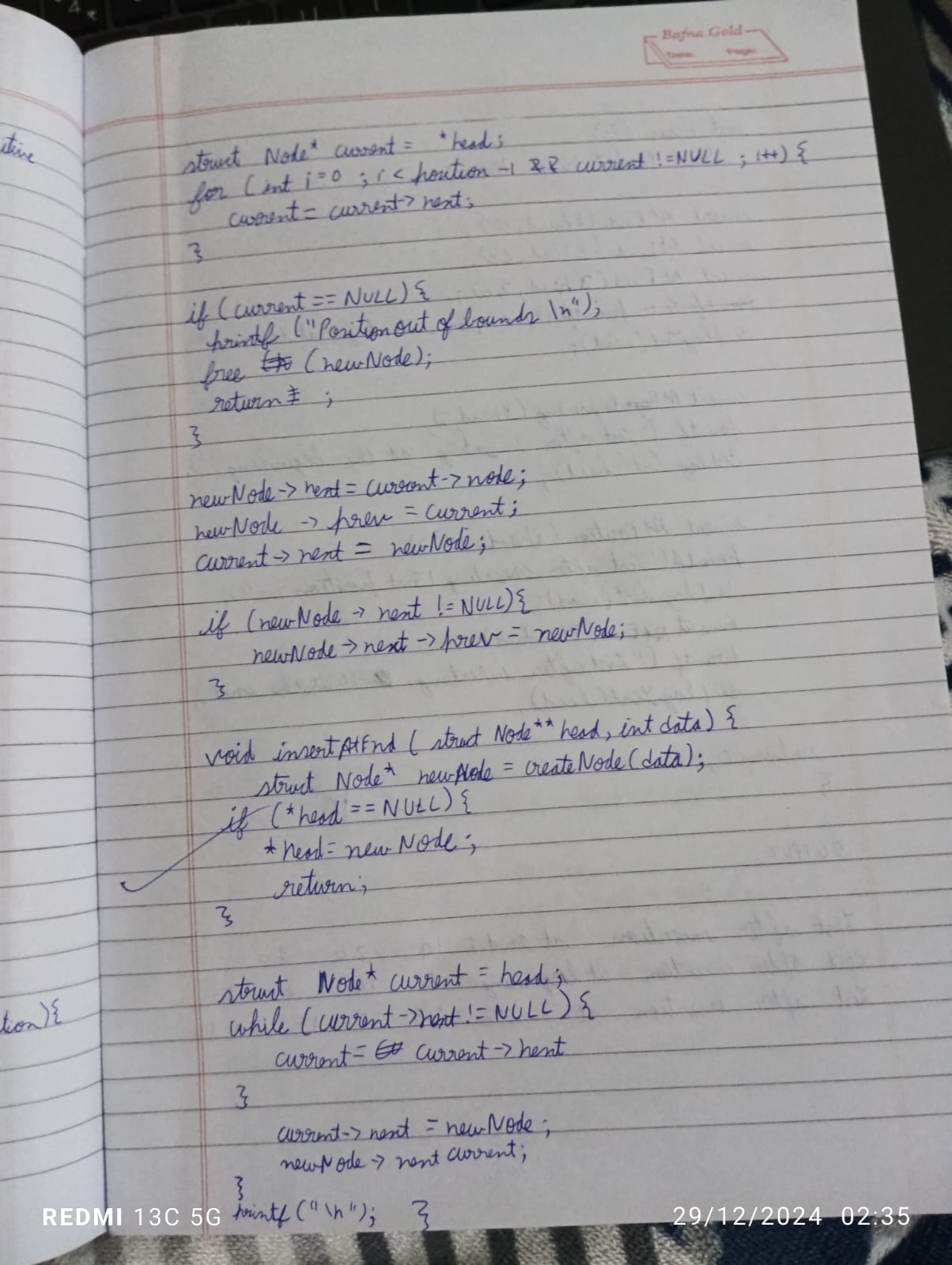
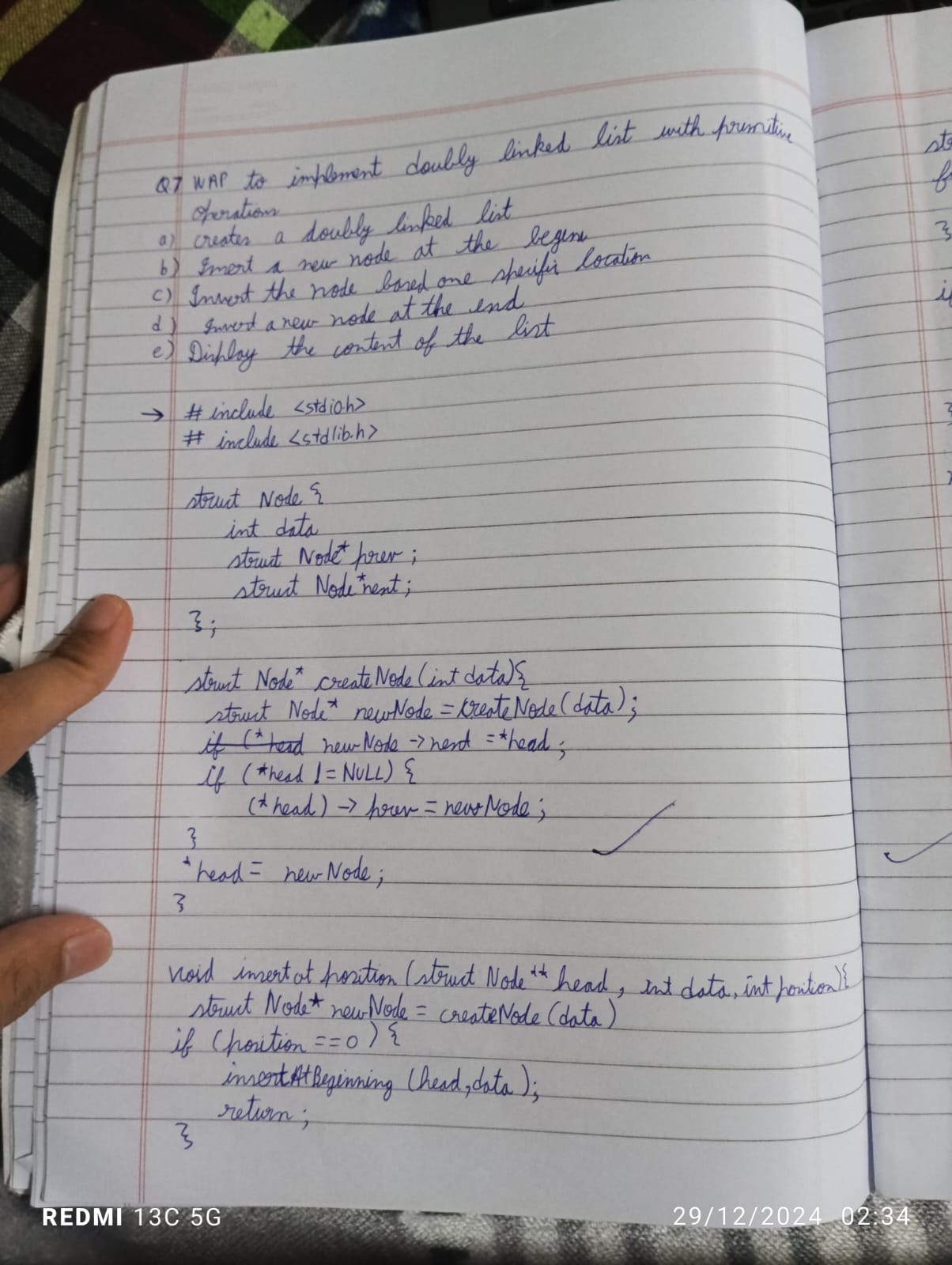
}

return 0;

}



8) 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



code:

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head = NULL;

struct node \*tail = NULL;

void add\_at\_begin(int x) {

struct node \*newnode;

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

newnode->data = x;

newnode->next = NULL;

newnode->prev = NULL;

if (head == NULL) {

head = tail = newnode;

} else {

newnode->next = head;

head->prev = newnode;

head = newnode;

}

}

void add\_at\_end(int x) {

struct node \*newnode;

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

newnode->data = x;

newnode->next = NULL;

newnode->prev = NULL;

if (head == NULL) {

head = tail = newnode;

} else {

newnode->prev = tail;

tail->next = newnode;

tail = newnode;

}

}

void insertpos(int x) {

struct node \*newnode, \*temp;

int pos;

printf("Enter the position: ");

scanf("%d", &pos);

if (pos == 1) {

add\_at\_begin(x);

} else {

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

newnode->data = x;

newnode->next = NULL;

newnode->prev = NULL;

temp = head;

for (int i = 1; i < pos - 1; i++) {

temp = temp->next;

if (temp == NULL) {

printf("There are less than %d nodes.\n", pos);

free(newnode);

return;

}

}

newnode->prev = temp;

newnode->next = temp->next;

if (temp->next != NULL) {

temp->next->prev = newnode;

}

temp->next = newnode;

if (newnode->next == NULL) {

tail = newnode;

}

}

}

void printlist() {

struct node \*temp;

temp = head;

if (temp == NULL) {

printf("The list is empty.\n");

} else {

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

}

int main() {

int choice, value;

while (1) {

printf("\nDoubly Linked List Menu:\n");

printf("1. Add at the beginning\n");

printf("2. Add at the end\n");

printf("3. Insert at a position\n");

printf("4. Print the list\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to add at the beginning: ");

scanf("%d", &value);

add\_at\_begin(value);

break;

case 2:

printf("Enter value to add at the end: ");

scanf("%d", &value);

add\_at\_end(value);

break;

case 3:

printf("Enter value to insert at a position: ");

scanf("%d", &value);

insertpos(value);

break;

case 4:

printlist();

break;

case 5:

printf("Exiting...\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

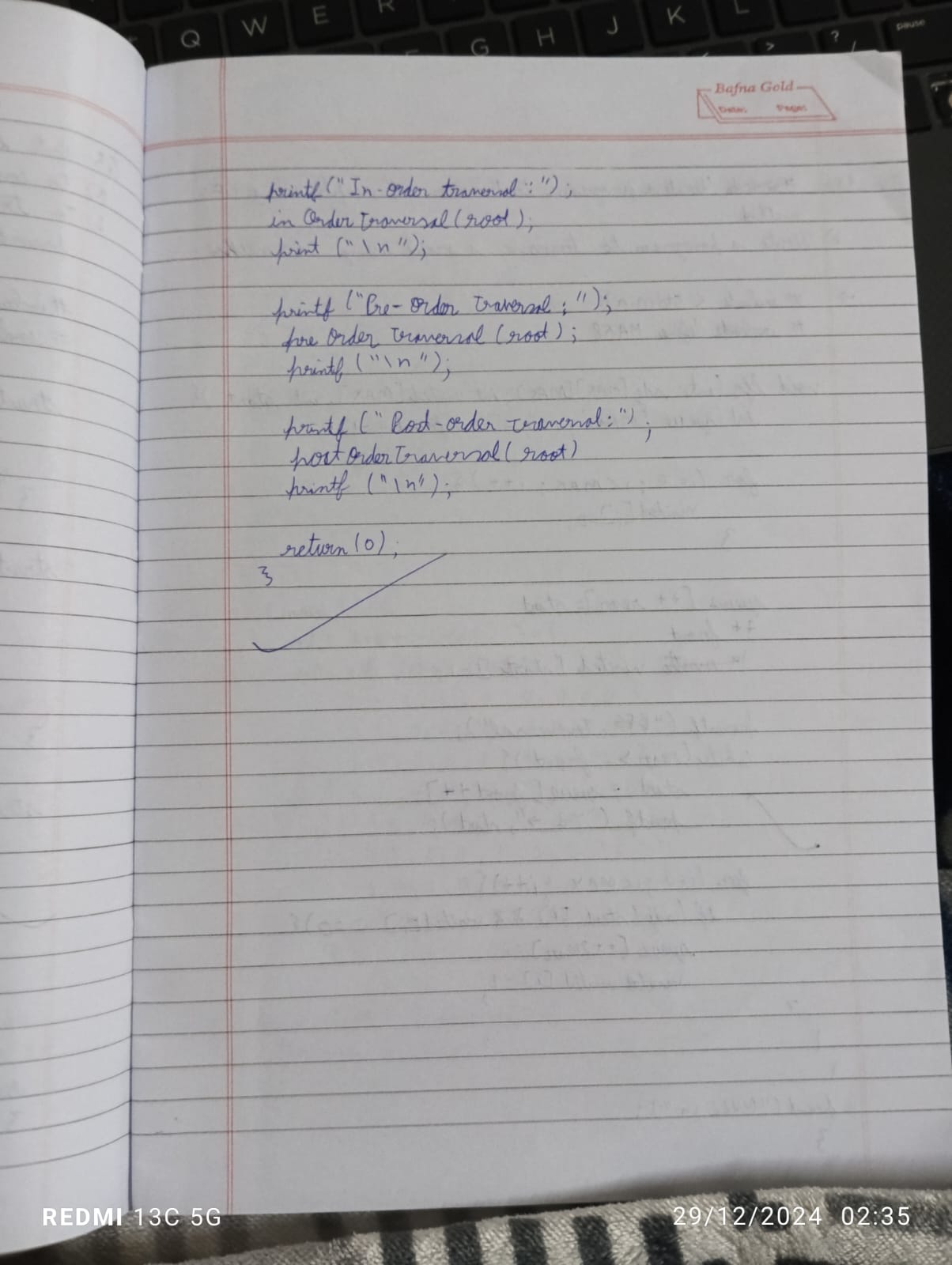
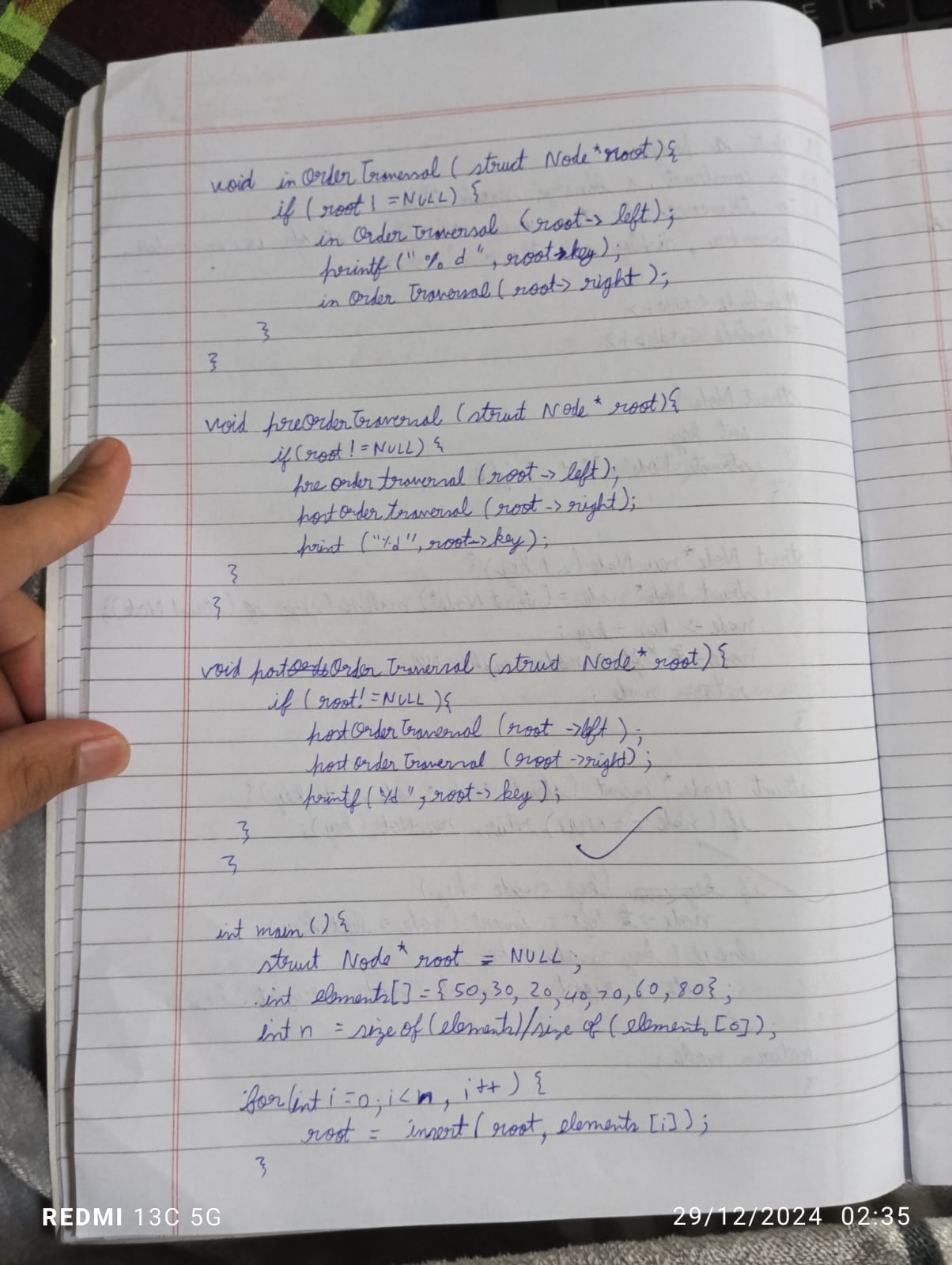
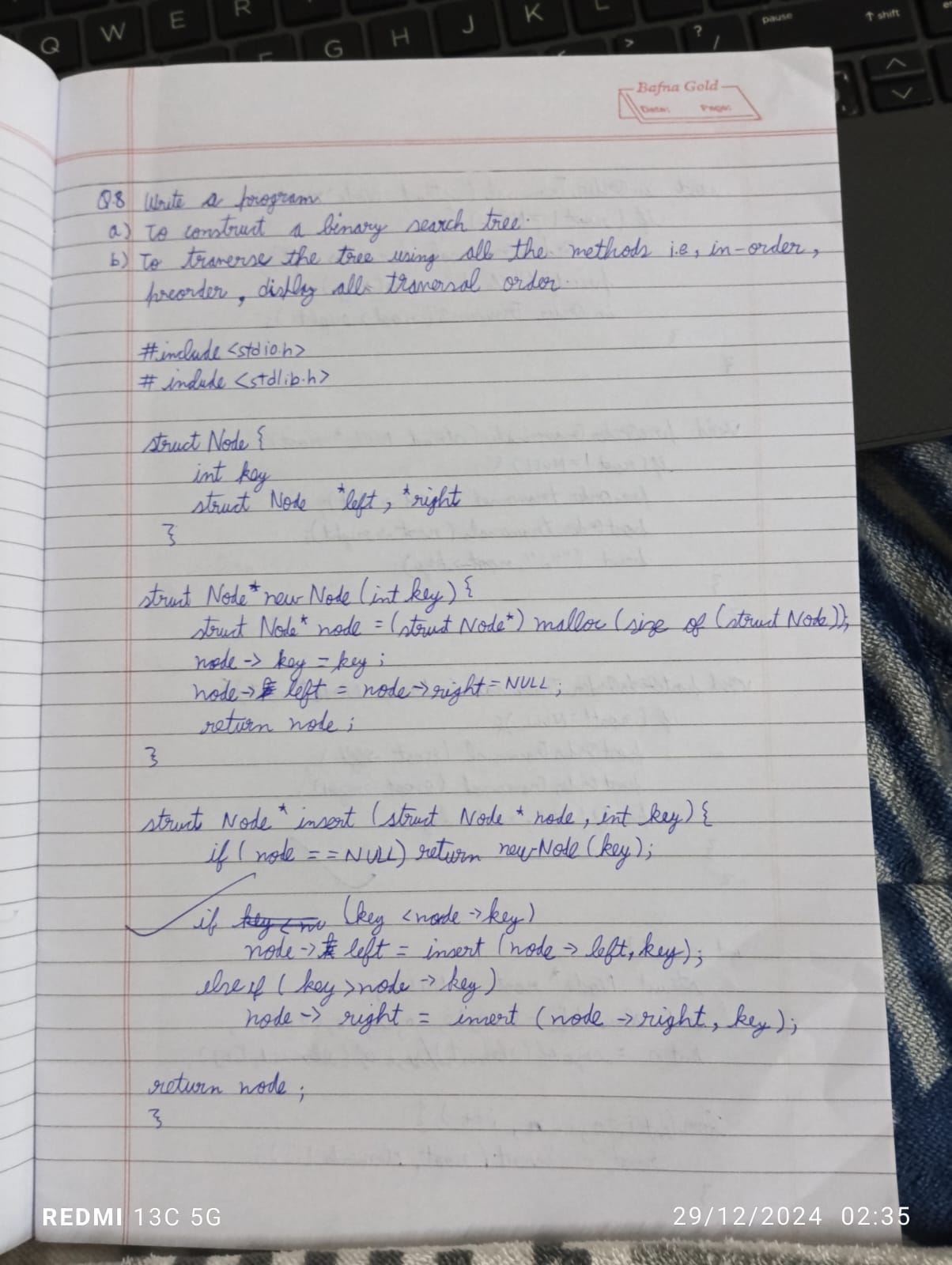
}

return 0;

}



9) 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



Code:

#include <stdio.h>

#include <stdlib.h>

struct Node {

int key;

struct Node \*left, \*right;

};

struct Node\* newNode(int key) {

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

node->key = key;

node->left = node->right = NULL;

return node;

}

struct Node\* insert(struct Node\* node, int key) {

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

void inOrderTraversal(struct Node\* root) {

if (root != NULL) {

inOrderTraversal(root->left);

printf("%d ", root->key);

inOrderTraversal(root->right);

}

}

void preOrderTraversal(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->key);

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

}

void postOrderTraversal(struct Node\* root) {

if (root != NULL) {

postOrderTraversal(root->left);

postOrderTraversal(root->right);

printf("%d ", root->key);

}

}

int main() {

struct Node\* root = NULL;

int elements[] = {50, 30, 20, 40, 70, 60, 80};

int n = sizeof(elements) / sizeof(elements[0]);

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

root = insert(root, elements[i]);

}

printf("In-order Traversal: ");

inOrderTraversal(root);

printf("\n");

printf("Pre-order Traversal: ");

preOrderTraversal(root);

printf("\n");

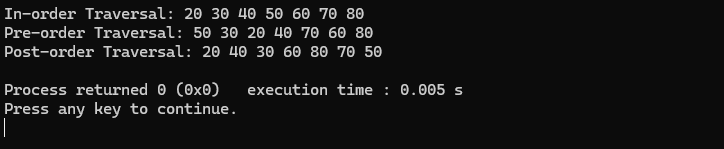
printf("Post-order Traversal: ");

postOrderTraversal(root);

printf("\n");

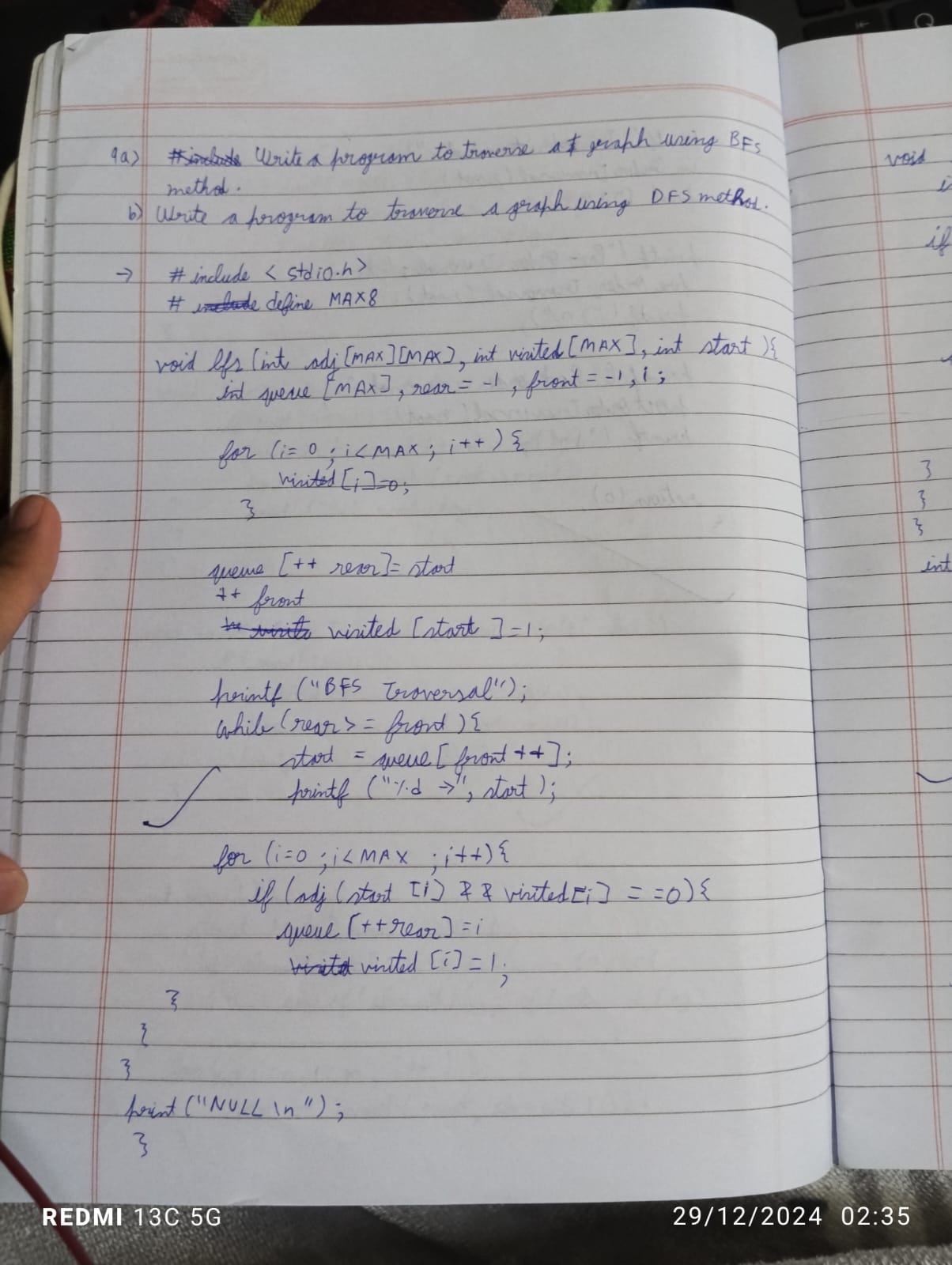
return 0;

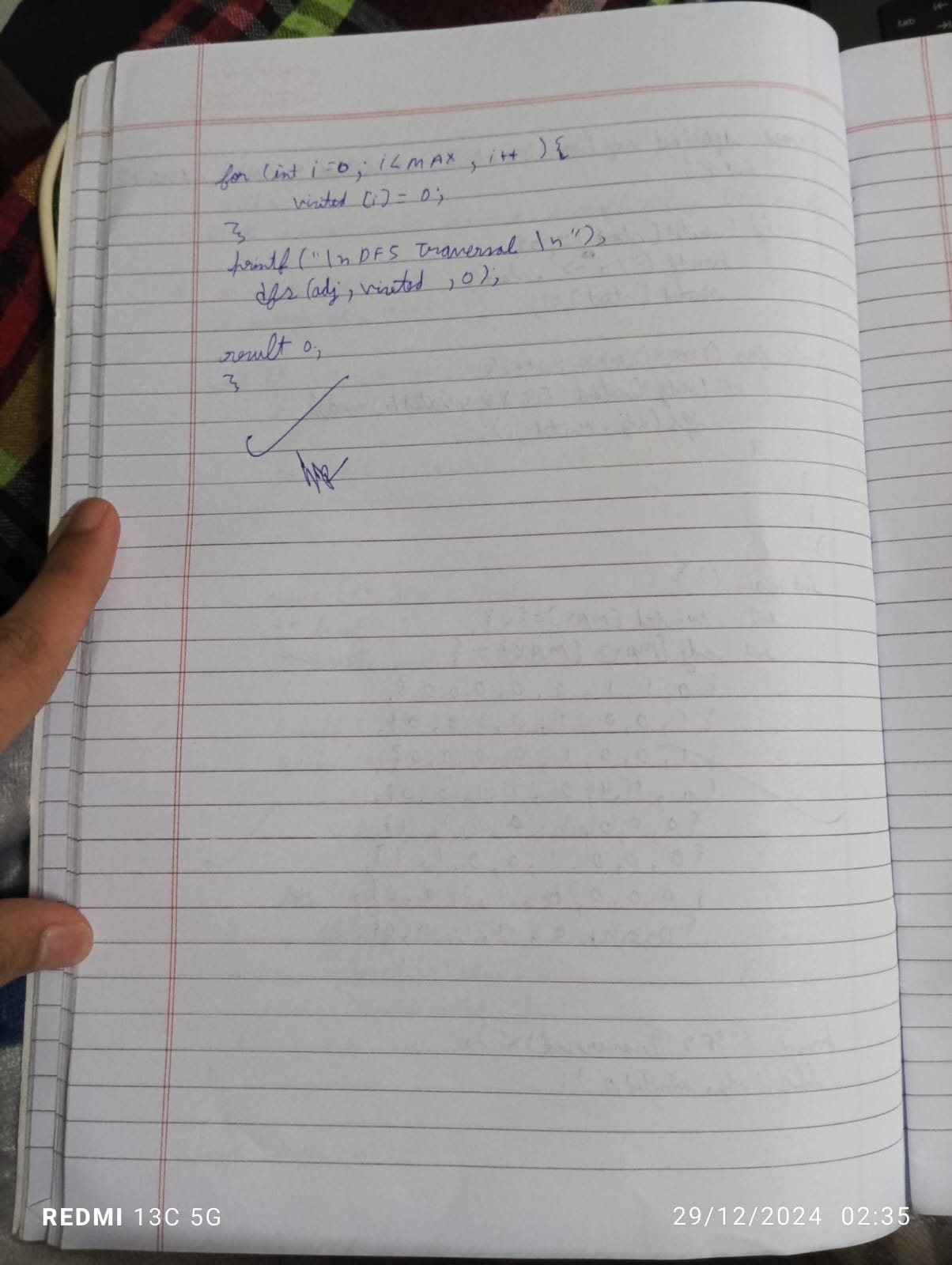
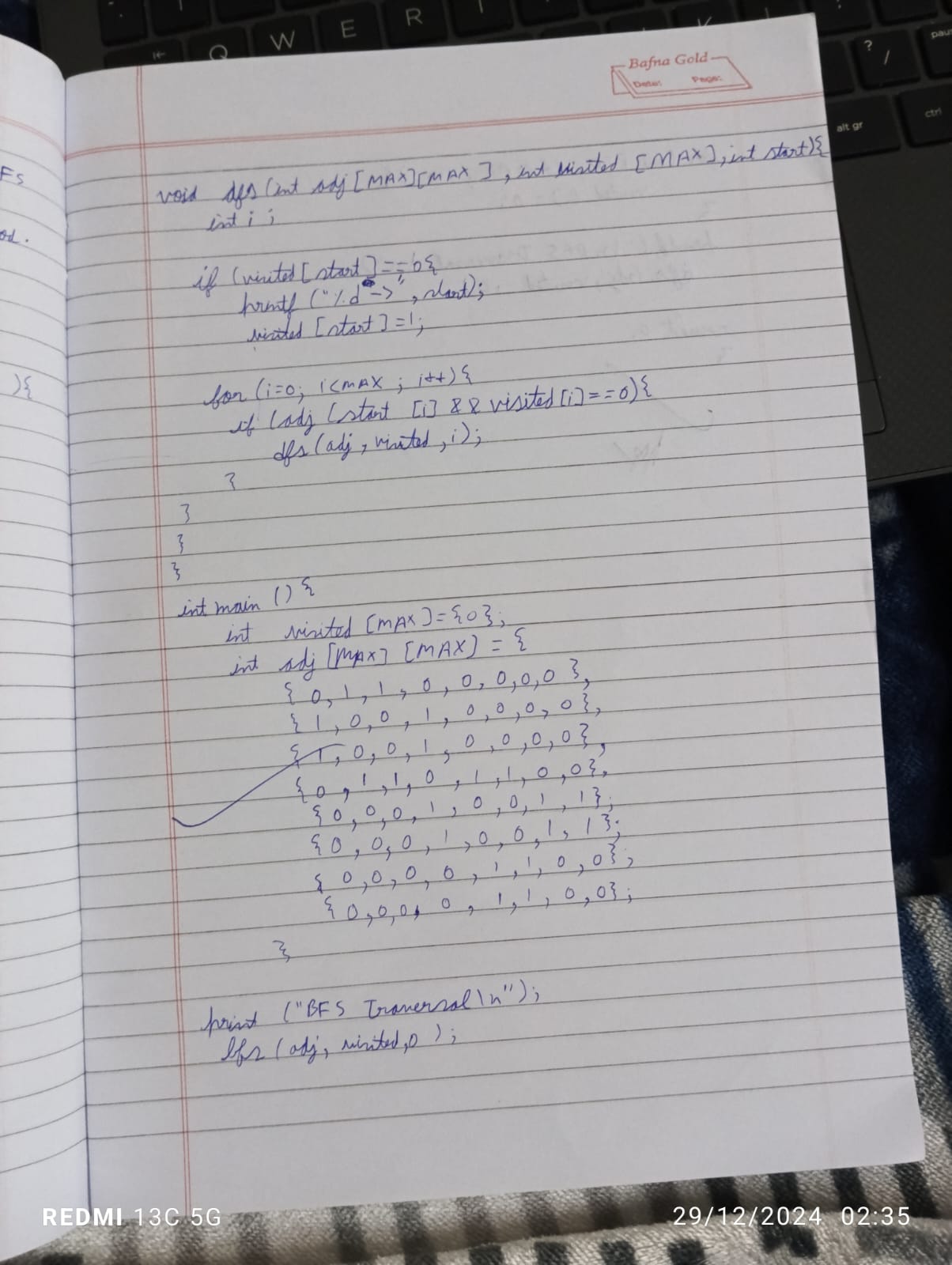
}



10(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.





Code:

#include <stdio.h>

#define MAX 8

void bfs(int adj[MAX][MAX], int visited[MAX], int start) {

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

for (i = 0; i < MAX; i++) {

visited[i] = 0;

}

queue[++rear] = start;

++front;

visited[start] = 1;

while (rear >= front) {

start = queue[front++];

printf("%d -> ", start);

for (i = 0; i < MAX; i++) {

if (adj[start][i] && visited[i] == 0) {

queue[++rear] = i;

visited[i] = 1;

}

}

}

printf("NULL\n");

}

int main() {

int visited[MAX] = {0};

int adj[MAX][MAX] = {

{0, 1, 1, 0, 0, 0, 0, 0},

{1, 0, 0, 1, 0, 0, 0, 0},

{1, 0, 0, 1, 0, 0, 0, 0},

{0, 1, 1, 0, 1, 1, 0, 0},

{0, 0, 0, 1, 0, 0, 1, 1},

{0, 0, 0, 1, 0, 0, 1, 1},

{0, 0, 0, 0, 1, 1, 0, 0},

{0, 0, 0, 0, 1, 1, 0, 0},

};

printf("BFS Traversal\n");

bfs(adj, visited, 0);

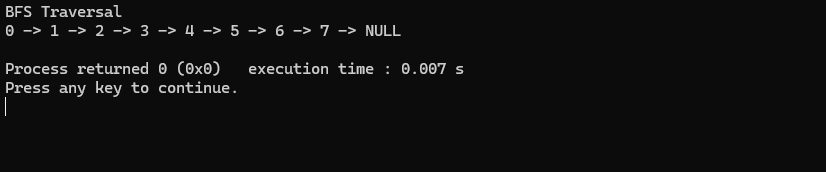
for (int i = 0; i < MAX; i++) {

visited[i] = 0;

}

return 0;

}

****

Code:

#include <stdio.h>

#define MAX 100

void dfs(int adj[MAX][MAX], int visited[MAX], int start, int n) {

int stack[MAX], top = -1;

int i;

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

visited[i] = 0;

}

stack[++top] = start;

visited[start] = 1;

printf("DFS Traversal: ");

while (top != -1) {

int current = stack[top--];

printf("%d -> ", current);

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

if (adj[current][i] == 1 && visited[i] == 0) {

stack[++top] = i;

visited[i] = 1;

}

}

}

printf("NULL\n");

}

int main() {

int n = 5;

int adj[MAX][MAX] = {

{0, 1, 1, 0, 0},

{1, 0, 0, 1, 1},

{1, 0, 0, 1, 0},

{0, 1, 1, 0, 1},

{0, 1, 0, 1, 0}

};

int visited[MAX];

int start = 0;

dfs(adj, visited, start, n);

return 0;

}

