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

Program 1

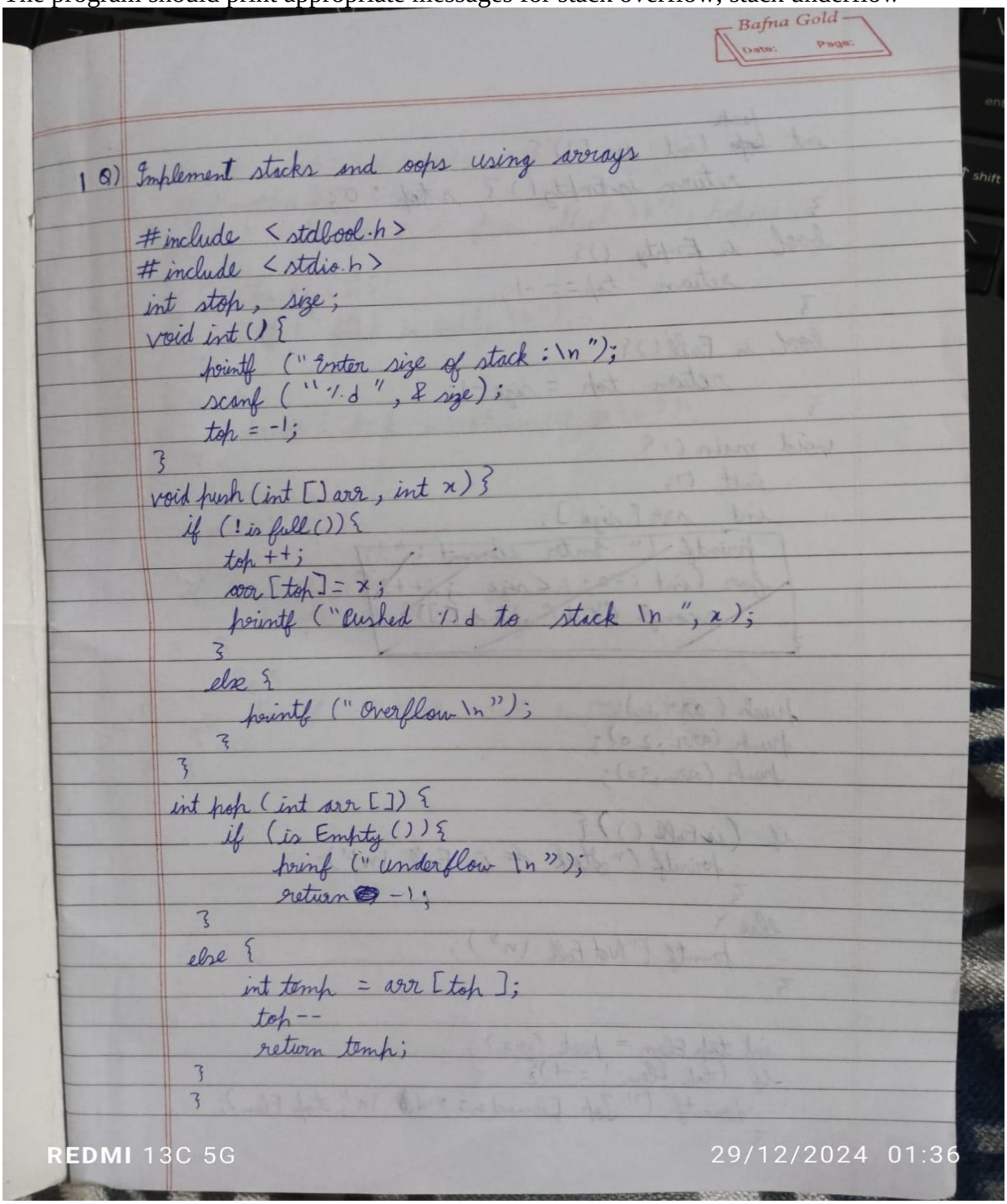
Write a program to simulate the working of stack using an array with the following:

a) Push

b) Pop

c) Display

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



```
10) Implement stacks and oops using arrays

#include <stdbool.h>
#include <stdio.h>
int stop, size;
void int () {
    printf ("Enter size of stack :\n");
    scanf ("%d", &size);
    top = -1;
}
void push (int [] arr, int x) {
  if (!is full()) {
     top ++;
     arr [top] = x;
     printf ("Pushed %d to stack \n", x);
    }
    else {
      printf ("Overflow \n");
    }
}
int pop (int arr []) {
    if (is Empty ()) {
        printf ("underflow \n");
        return -1;
  }
  else {
      int temp = arr [top];
       top--
       return temp;
  }
}
```

Bafna Gold
Date: Page:
REDMI 13C 5G 29/12/2024 01:36

```
int peek (int arr[]) {
    return isEmpty() ? s.top : 0;
}
bool isEmpty (){
    return top == -1;
}
bool isFull (){
    return top = size -1;
}
void main (){
    int ();
    int arr[size];
    printf (" enter element :");
    for (int i=0; i<size ;i++){
        scanf ("%d", &s[i]);
}

push (arr,10);
push (arr,20);
push (arr,30);

if (isFull ()){
    printf ("Stack is Full \n");
}
else{
    printf ("Not Full \n");
}

int topElem = peek (arr);
if (topElem != -1){
    printf ("Top Element is: %d \n", topElem);
}
```

REDMI 13C 5G 29/12/2024 01:37

```
int poppedVal = pop(arr);
if (poppedVal != -1){
printf ("Popped %d from stack \n", poppedVal);
}
if (isEmpty()){
    printf ("stack is empty \n");
}
else {
    printf ("stack is not empty \n");
}
push(arr)
push(arr);

if (isEmpty()){
    printf ("stack is empty \n");
}
else {
    printf ("stack is not empty \n");
}
}
```

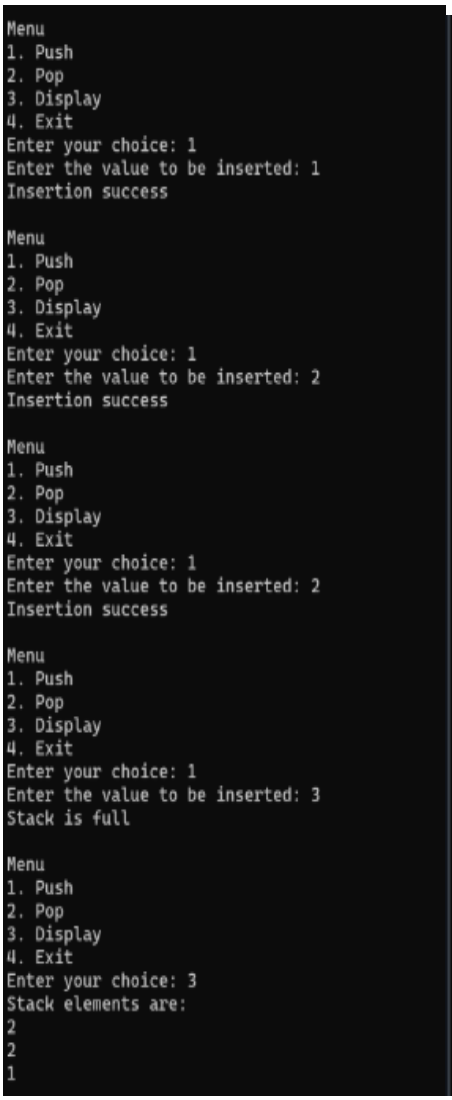
Seen

REDMI 13C 5G 29/12/2024 01:37

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");
      }
   }
```

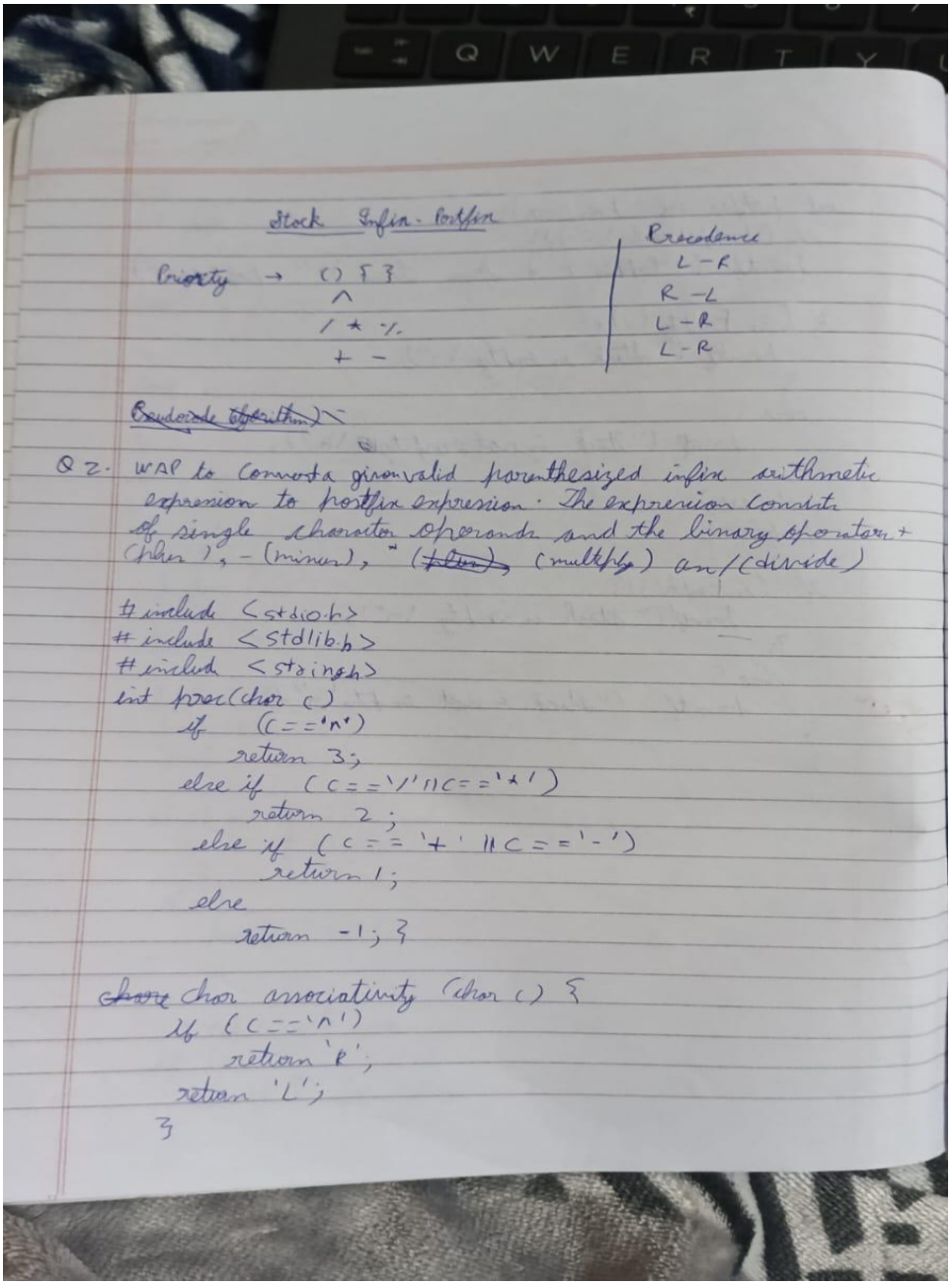
```
    return 0;
}

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]);
        }
    }
}
```



```
Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 1
Insertion success

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 2
Insertion success

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 2
Insertion success

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 3
Stack is full

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements are:
2
2
1
```

```
Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted 2

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted 2

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Deleted 1

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Stack is empty

Menu
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 28.584 s
Press any key to continue.
```

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)



Stack Infix - Postfix

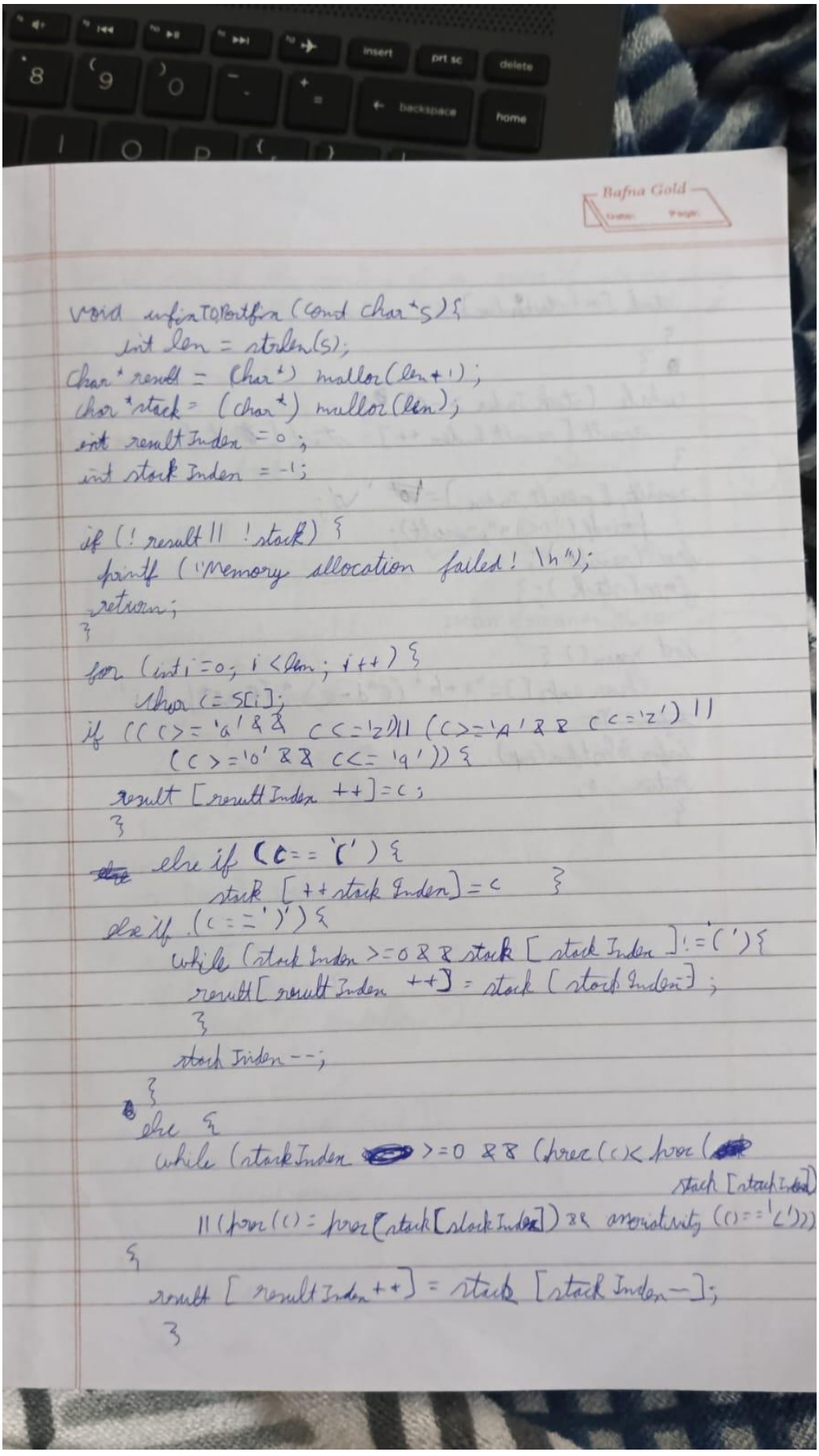
Priority → () { }
^
/ * %
+ -

Precedence
L - R
R - L
L - R
L - R

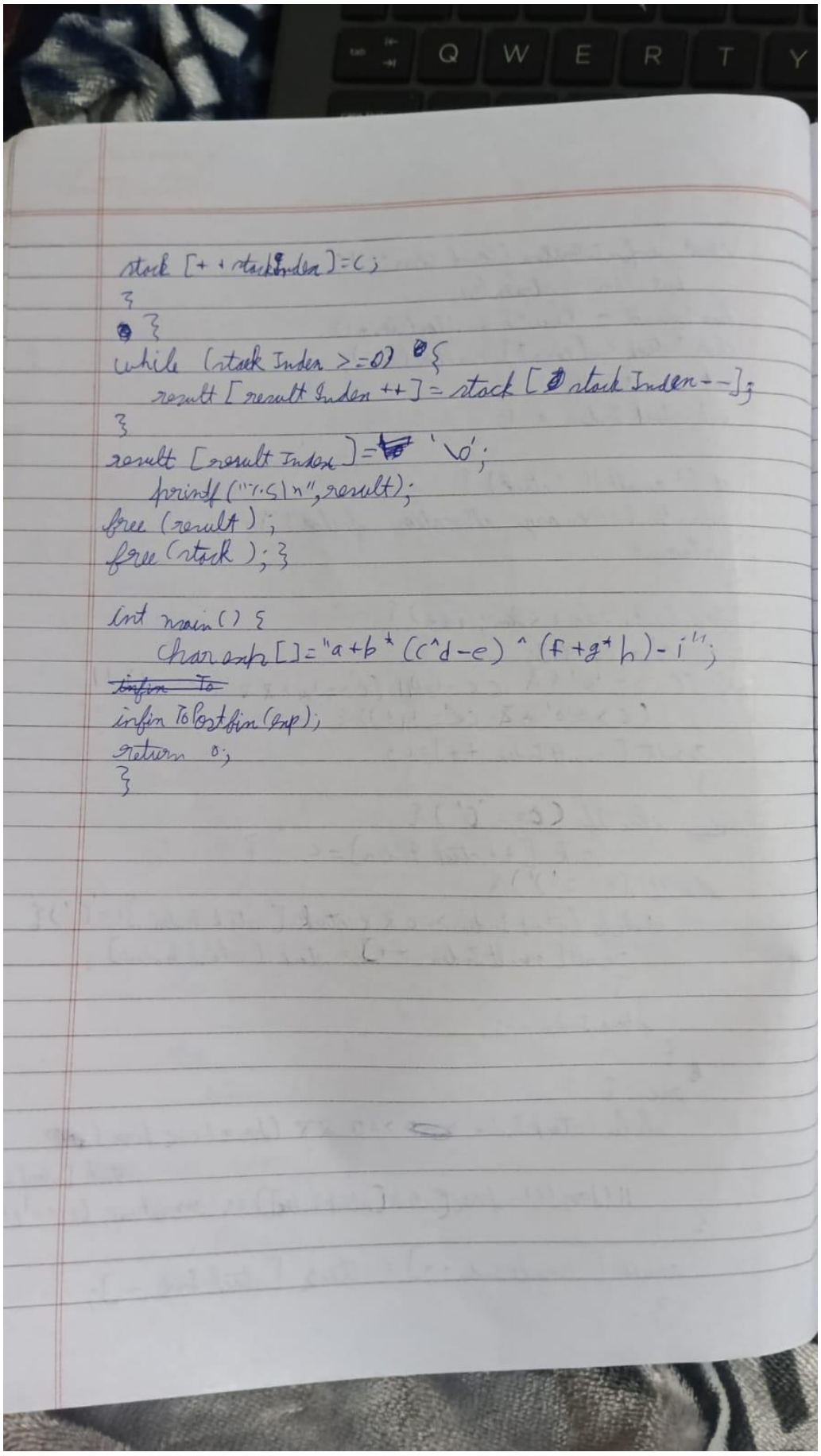
~~Evaluate Operation~~

Q2. WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <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 *s){
    int len = strlen(s);
char *result = (char*) malloc(len+1);
char *stack = (char*) malloc(len);
int resultIndex = 0;
int stackIndex = -1;

if (!result || !stack) {
 printf ("Memory allocation failed! \n");
 return;
}
for (int i=0; i<len; i++) {
    char c = s[i];
if ((c>='a' && c<='z') || (c>='A' && c<='z') ||
      (c>='0' && c<='9')) {
 result [resultIndex ++] = c;
 }
 else if (c=='(') {
      stack [++stackIndex] = c   }
 else if (c==')') {
    while (stackIndex >=0 && stack [stackIndex] != '(') {
     result[resultIndex ++] = stack [stackIndex];
      }
     stackIndex --;
  }
  else {
    while (stackIndex >=0 && (prec(c)< prec(
                                    stack [stackIndex]
     || (prec(c) = prec(stack[stackIndex]) && associativity(c)=='L'))
{
 result [resultIndex++] = 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;
}
```

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");
```

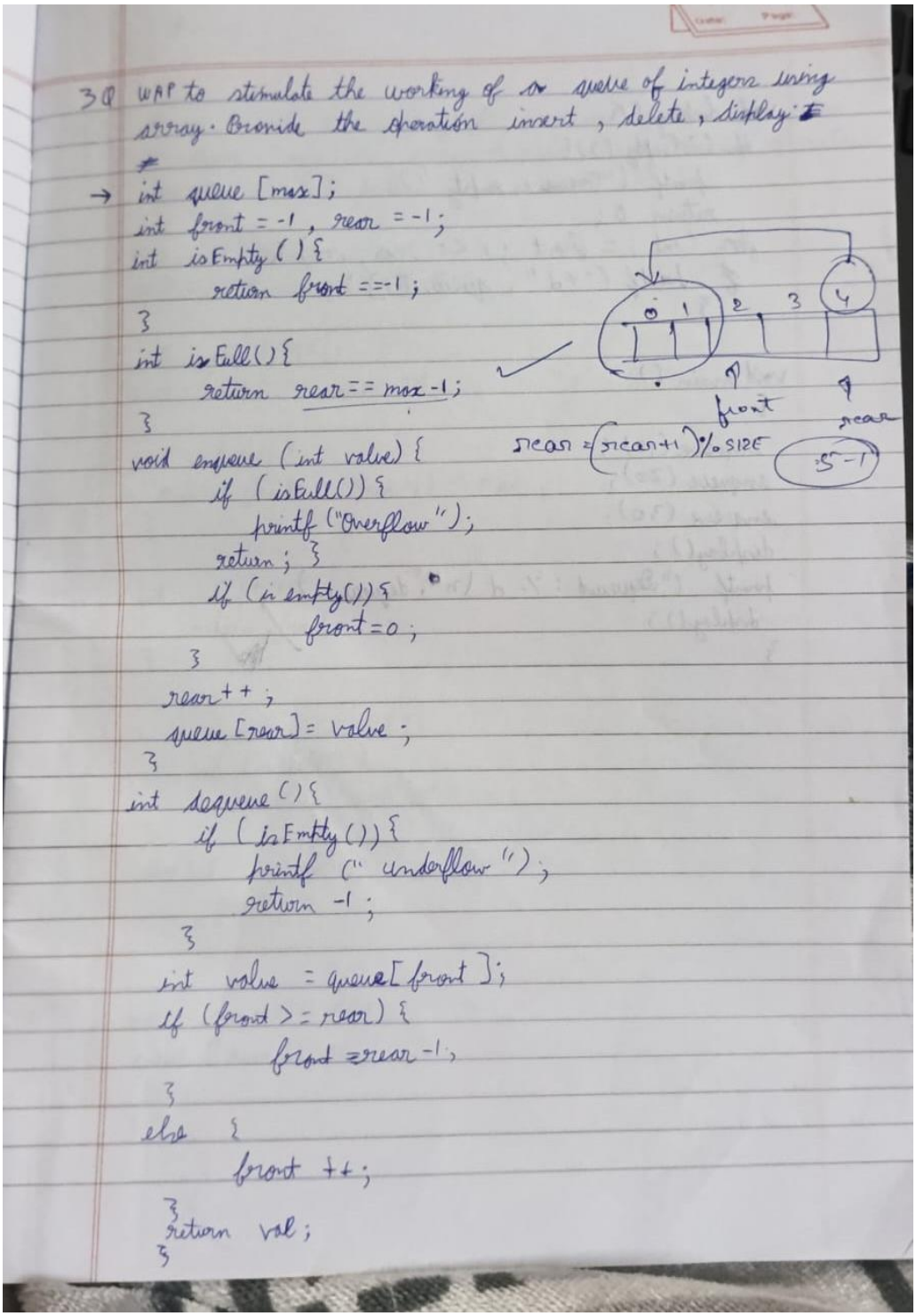
```
    } else {
        printf("Deleted %d\n", queue[front]);
        front++;
        if (is_empty()) {
            front = rear = -1;
        }
    }
}

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;
}
```

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

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

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

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter a value to insert: 4
Queue overflow! Cannot insert 4

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue: 1 2 3
```

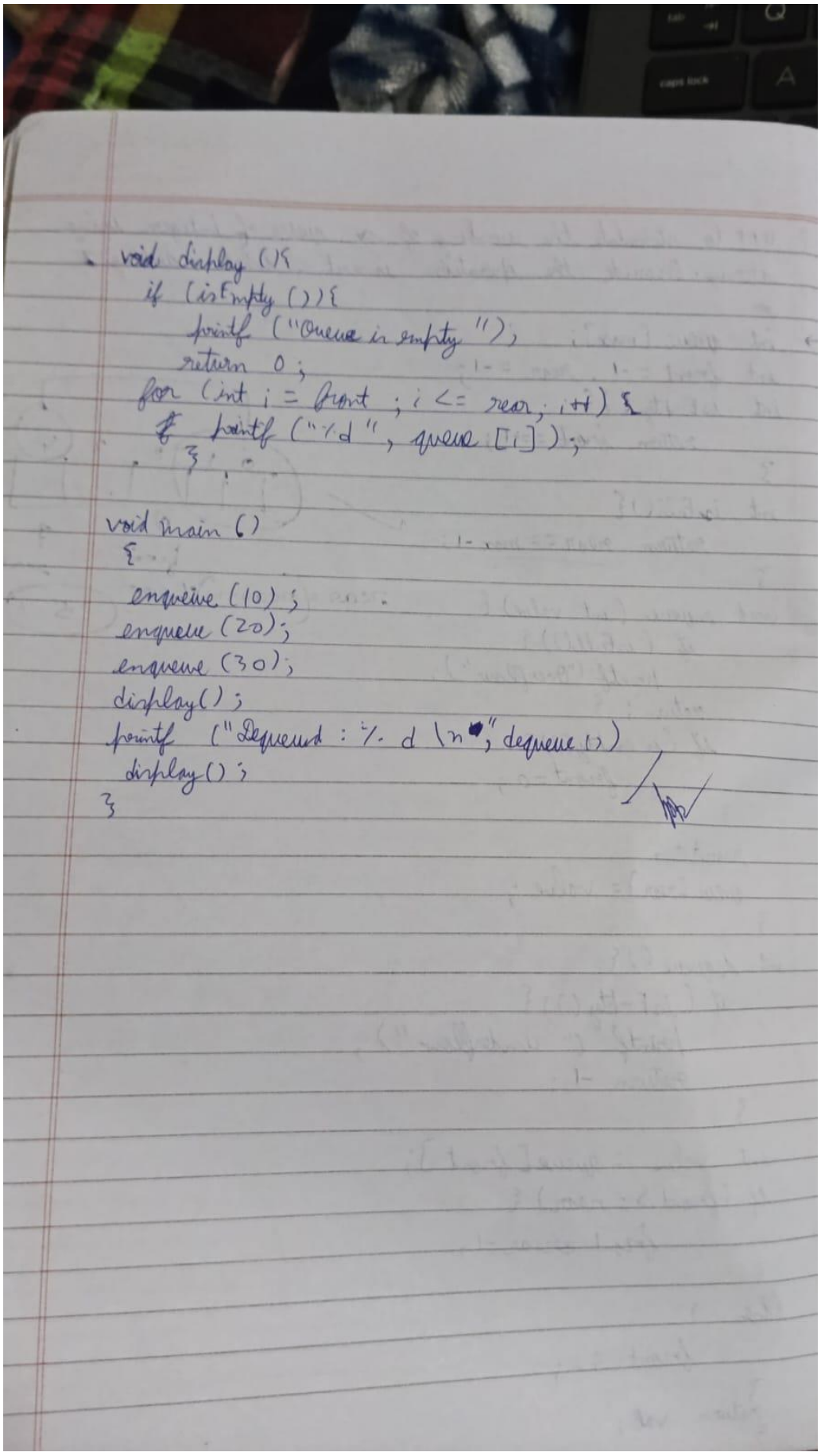
```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 1

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

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

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue underflow! Cannot delete

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 7
Invalid choice! Please try again.
```

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



3Q WAP to simulate the working of a queue of integers using array. Provide the operation insert, delete, display

```
int queue [max];
int front = -1, rear = -1;
int isEmpty () {
    return front == -1;
}
int isFull () {
    return rear == max - 1;
}
void enqueue (int value) {
    if (isFull()) {
        printf ("Overflow");
        return; }
    if (isempty()) {
        front = 0;
    }
    rear++;
    queue [rear] = value;
}
int dequeue () {
    if (isEmpty()) {
        printf ("underflow");
        return -1;
    }
    int value = queue [front];
    if (front >= rear) {
        front = rear - 1;
    }
    else {
        front ++;
    }
    return val;
}
```



```
void display ()
  if (isEmpty ()){
     printf ("Queue is empty");
     return 0;
  for (int i = front ; i <= rear; i++) {
     printf ("%d", queue [i]);
    };

void main ()
 {
  enqueue (10);
  enqueue (20);
  enqueue (30);
  display();
  printf ("Dequeued : %. d \n", dequeue ())
  display();
}
```

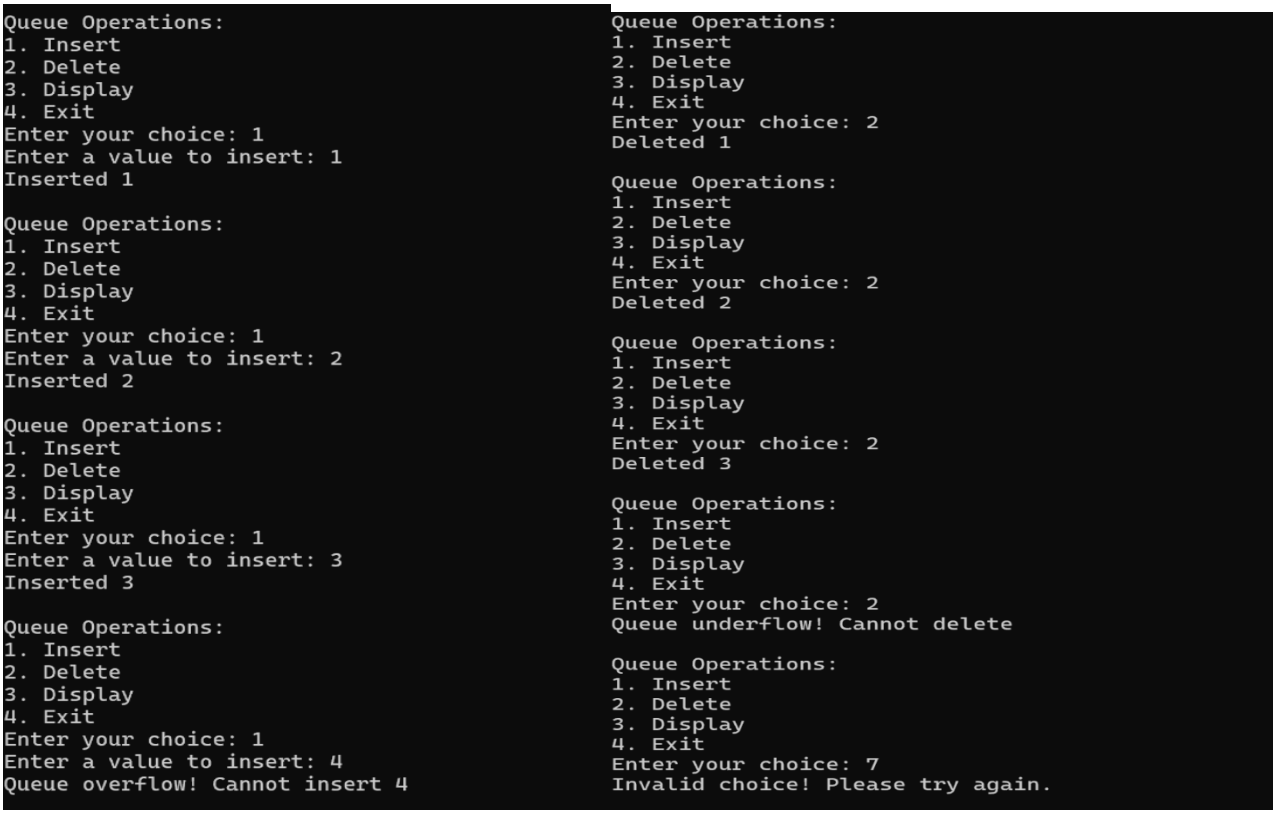
Code:

```
#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;
```

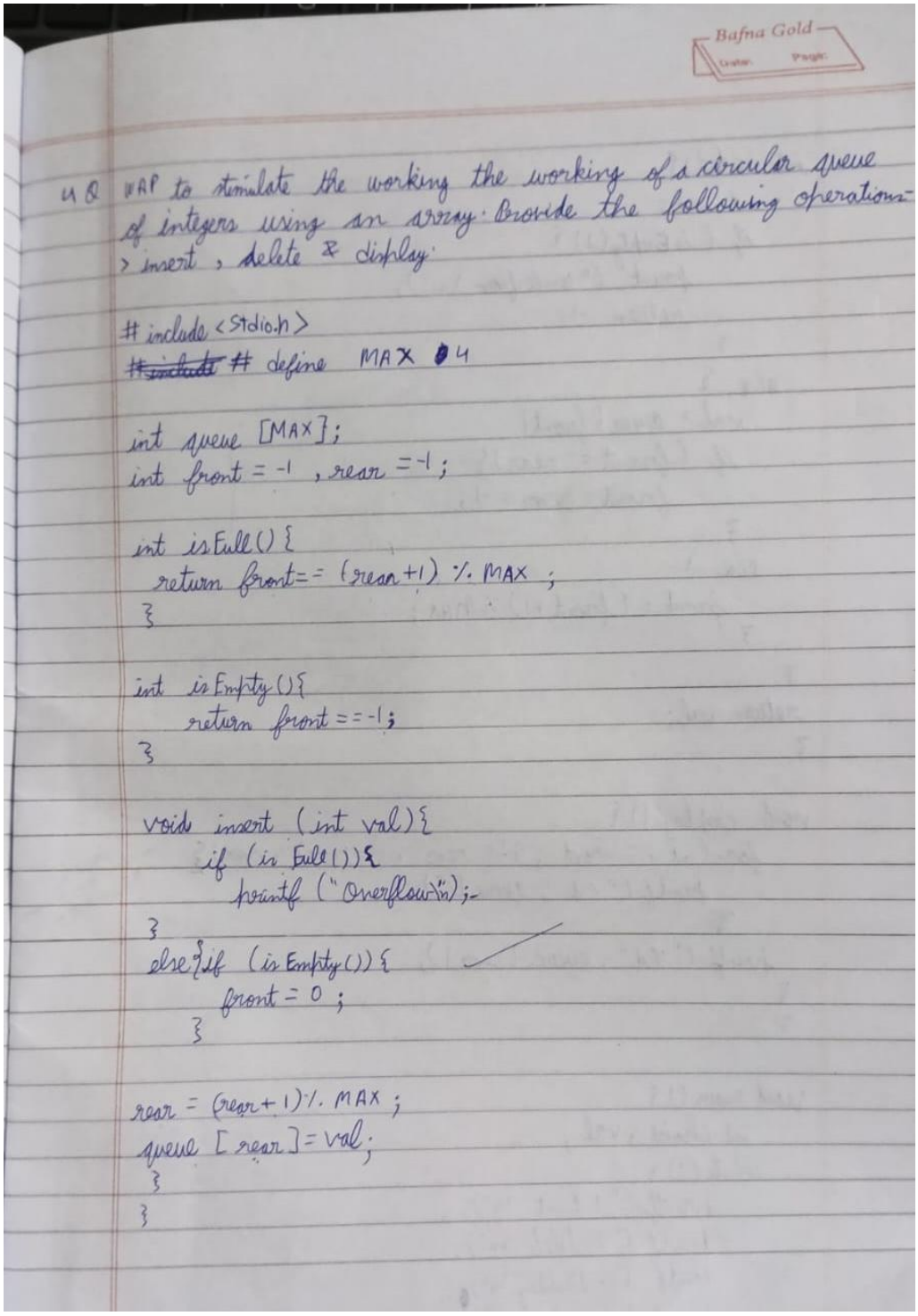
```
        }
        queue[++rear] = value;
        printf("Inserted %d\n", value);
    }
}

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;
```

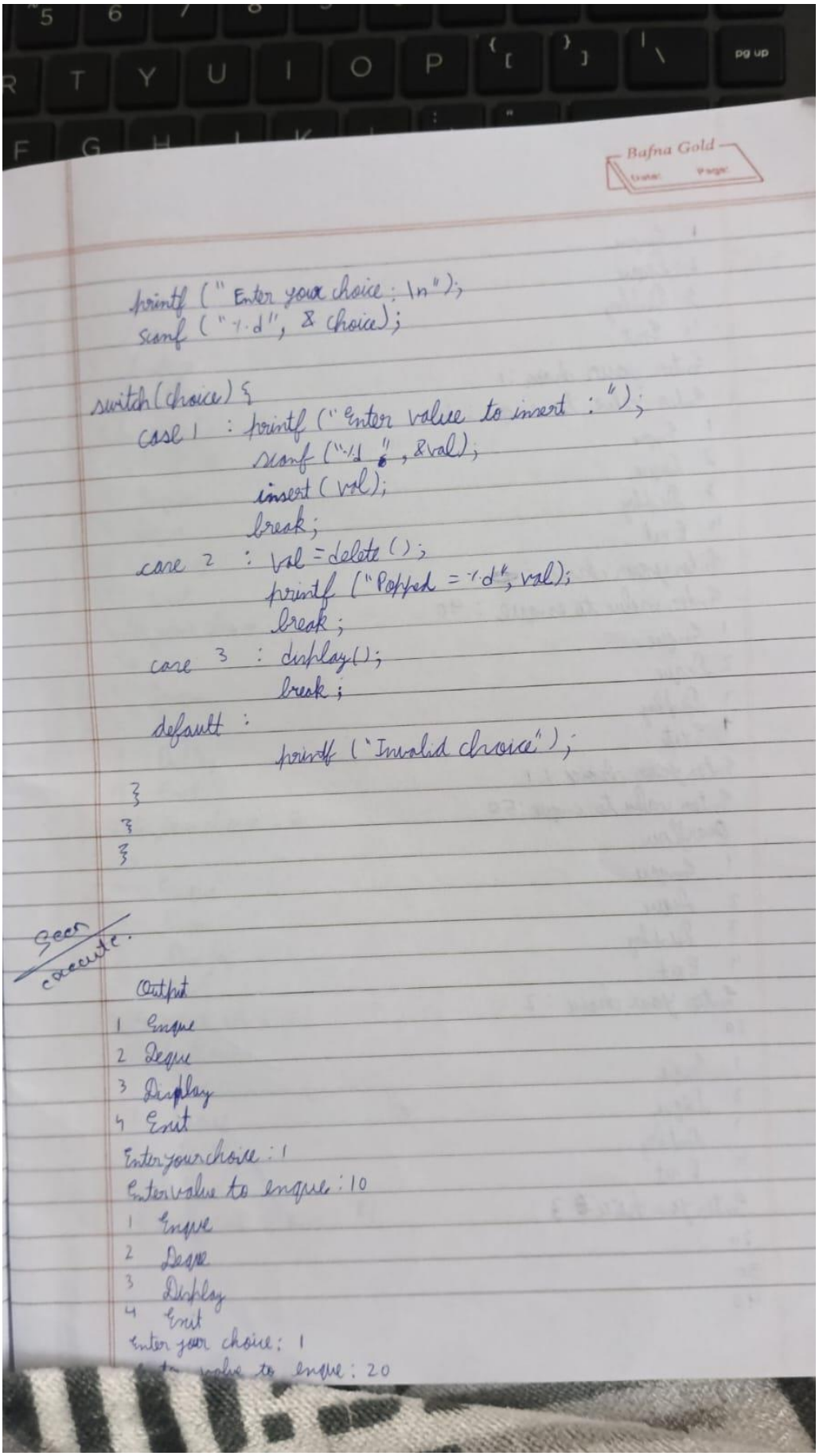
```
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n")}
    return 0;
}
```



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



```
4Q WAP to simulate the working the working of a circular queue
of integers using an array. Provide the following operations-
> insert , delete & display

#include <stdio.h>
#~~include~~ #define MAX 4

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

int isFull() {
  return front == (rear+1) % MAX ;
}

int isEmpty(){
    return front == -1;
}

void insert (int val){
    if (isFull()){
      printf ("Overflow\n");
}
else{if (isEmpty()) {
      front = 0 ;
    }

rear = (rear+1) % MAX ;
queue [rear] = val;
}
}
```

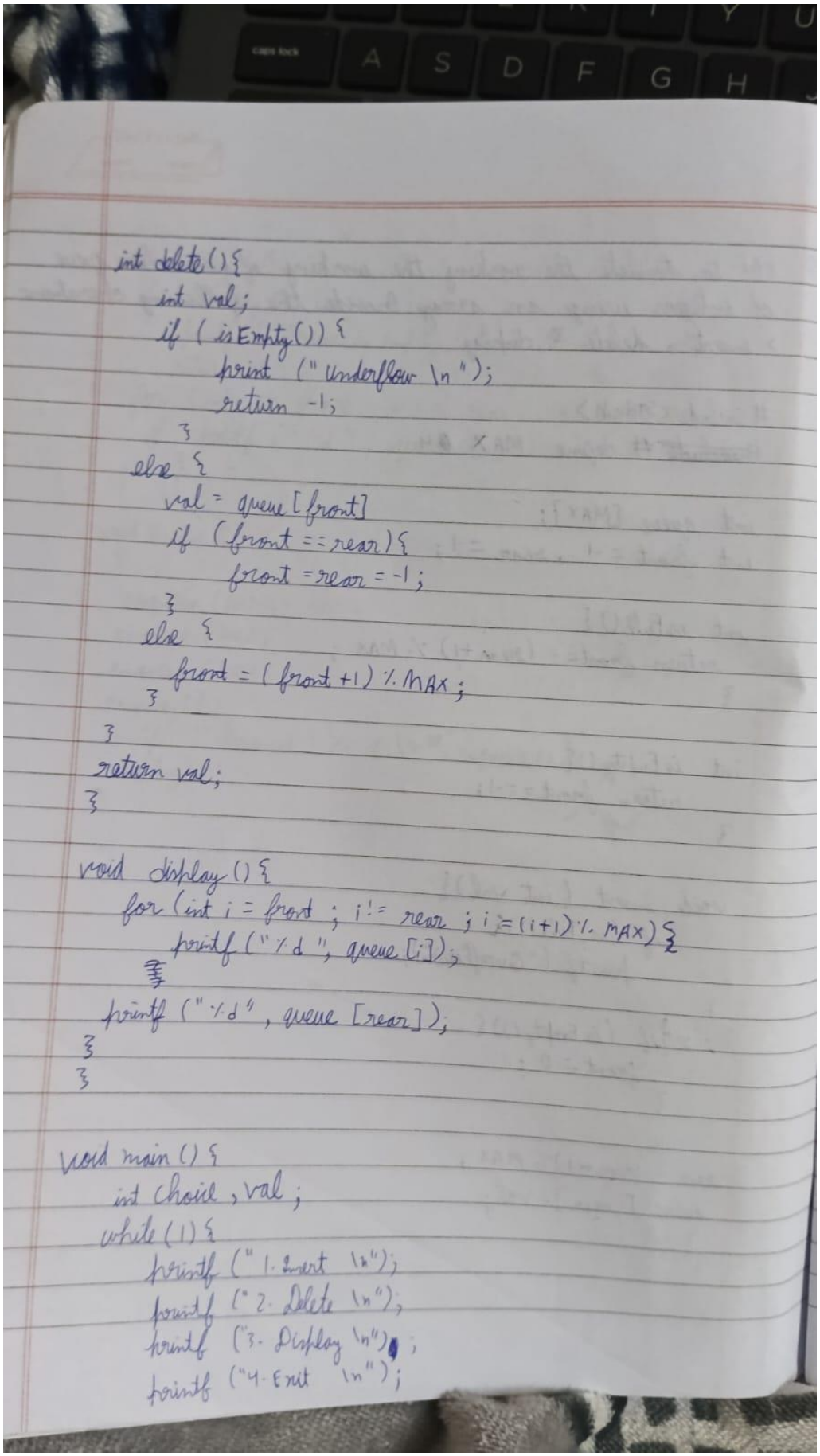


```
printf ("Enter your choice: \n");
scanf ("%d", &choice);

switch(choice) {
    case 1 : printf ("Enter value to insert : ");
             scanf ("%d", &val);
             insert (val);
             break;
    case 2 : val = delete ();
             printf ("Popped = %d", val);
             break;
    case 3 : display();
             break;
    default :
             printf ("Invalid choice");
}
}
}
```

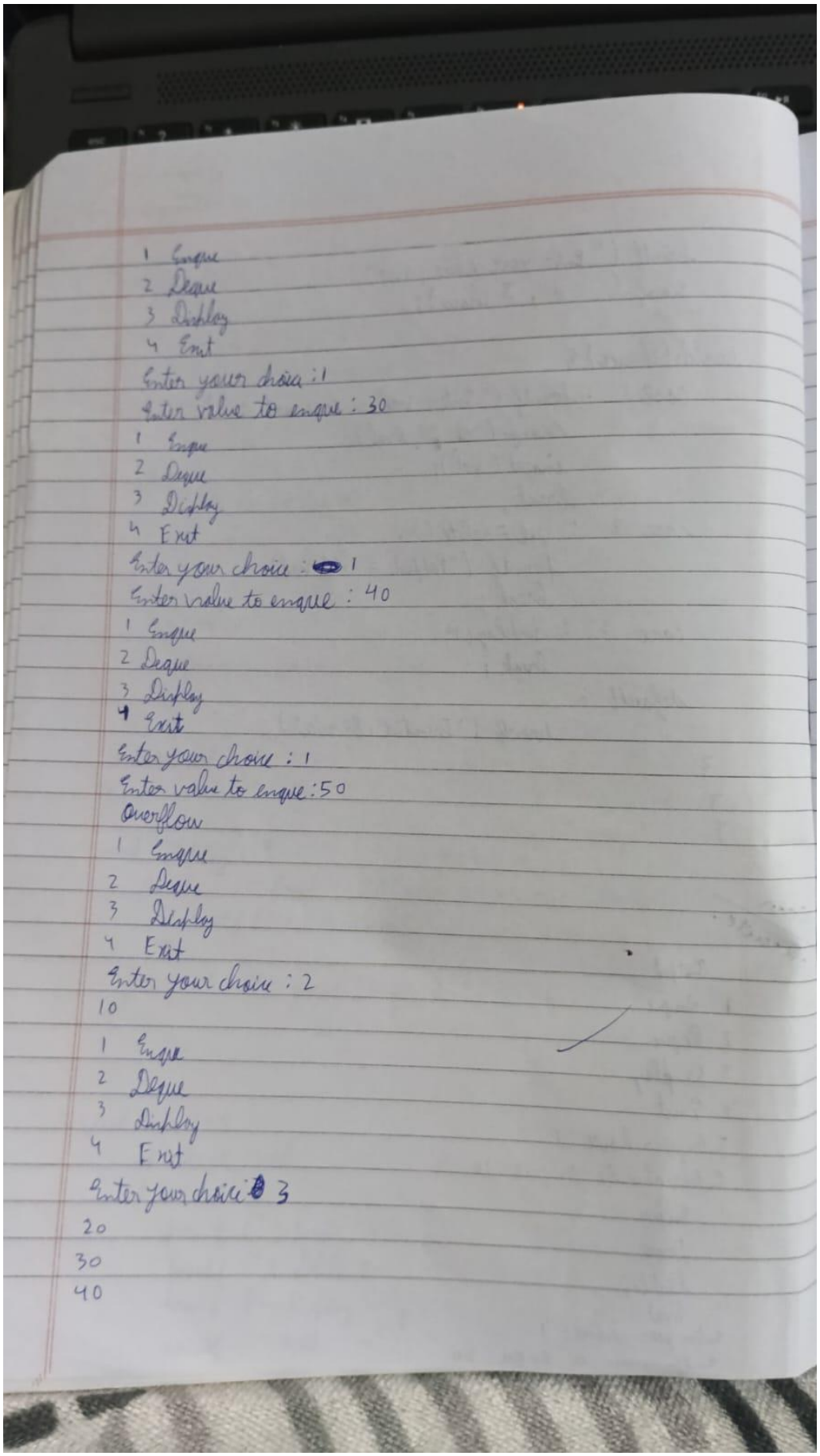
Seen / execute.

Output

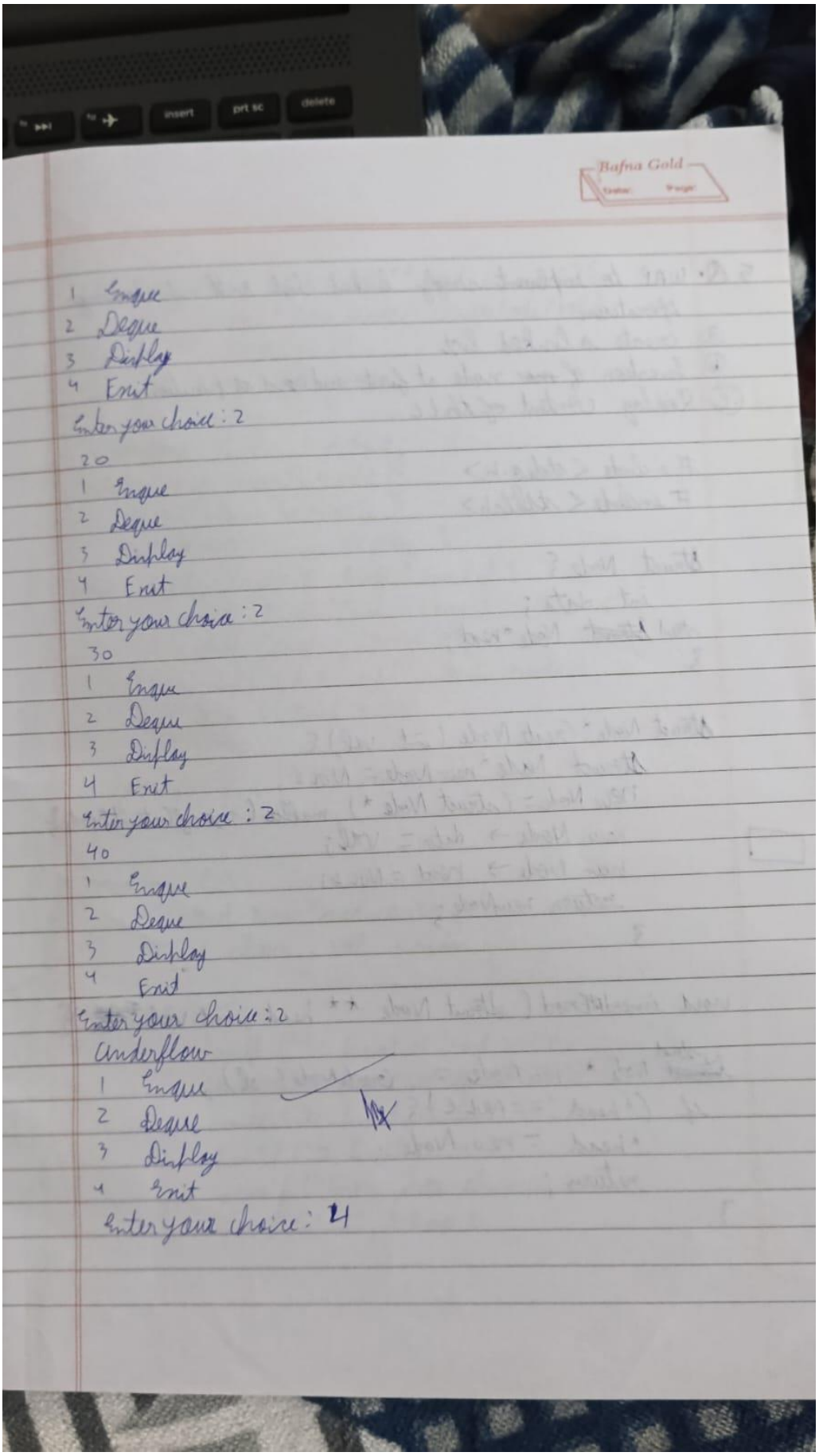
```
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 1
Enter value to enque: 10
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 1
Enter value to enque: 20
```



```
int delete(){
    int val;
    if (isEmpty()) {
        print ("Underflow \n");
        return -1;
    }
  else {
    val = queue[front]
    if (front == rear){
        front = rear = -1;
    }
    else {
      front = (front+1) % MAX;
    }
}
return val;
}

void display (){
    for (int i = front ; i != rear ; i = (i+1) % MAX) {
        printf ("%d", queue[i]);
    }
  printf ("%d", queue[rear]);
}
}


void main (){
    int choice, val;
    while (1) {
        printf ("1. Insert \n");
        printf ("2. Delete \n");
        printf ("3. Display \n");
        printf ("4. Exit \n");
```



1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 1
Enter value to enque: 30
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 1
Enter value to enque: 40
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 1
Enter value to enque: 50
Overflow
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 2
10
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 3
20
30
40



1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 2
20
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 2
30
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 2
40
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 2
Underflow
1 Enque
2 Deque
3 Display
4 Exit
Enter your choice: 4

Code:

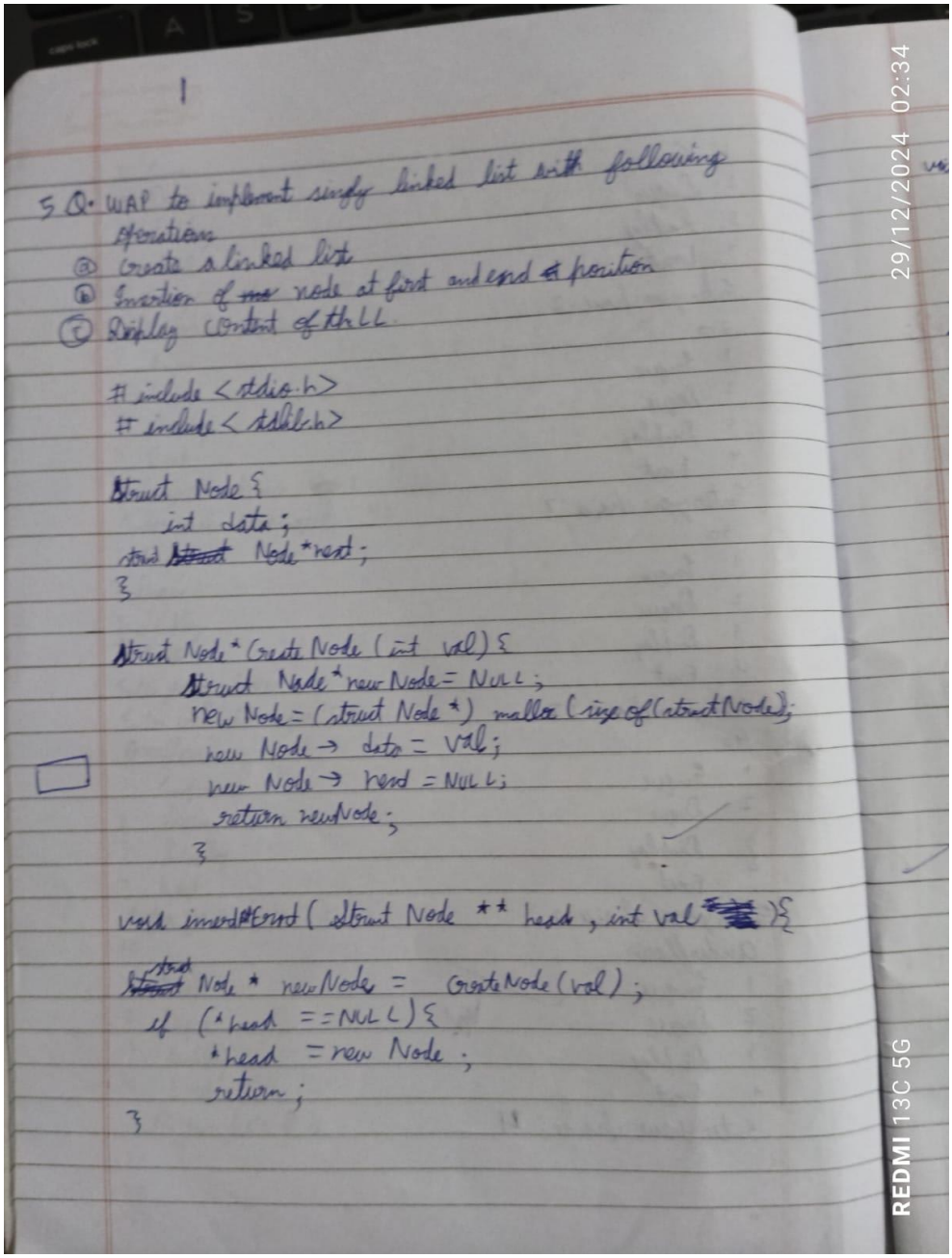
```
#include <stdio.h>
#define MAX 3

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;
   }
```

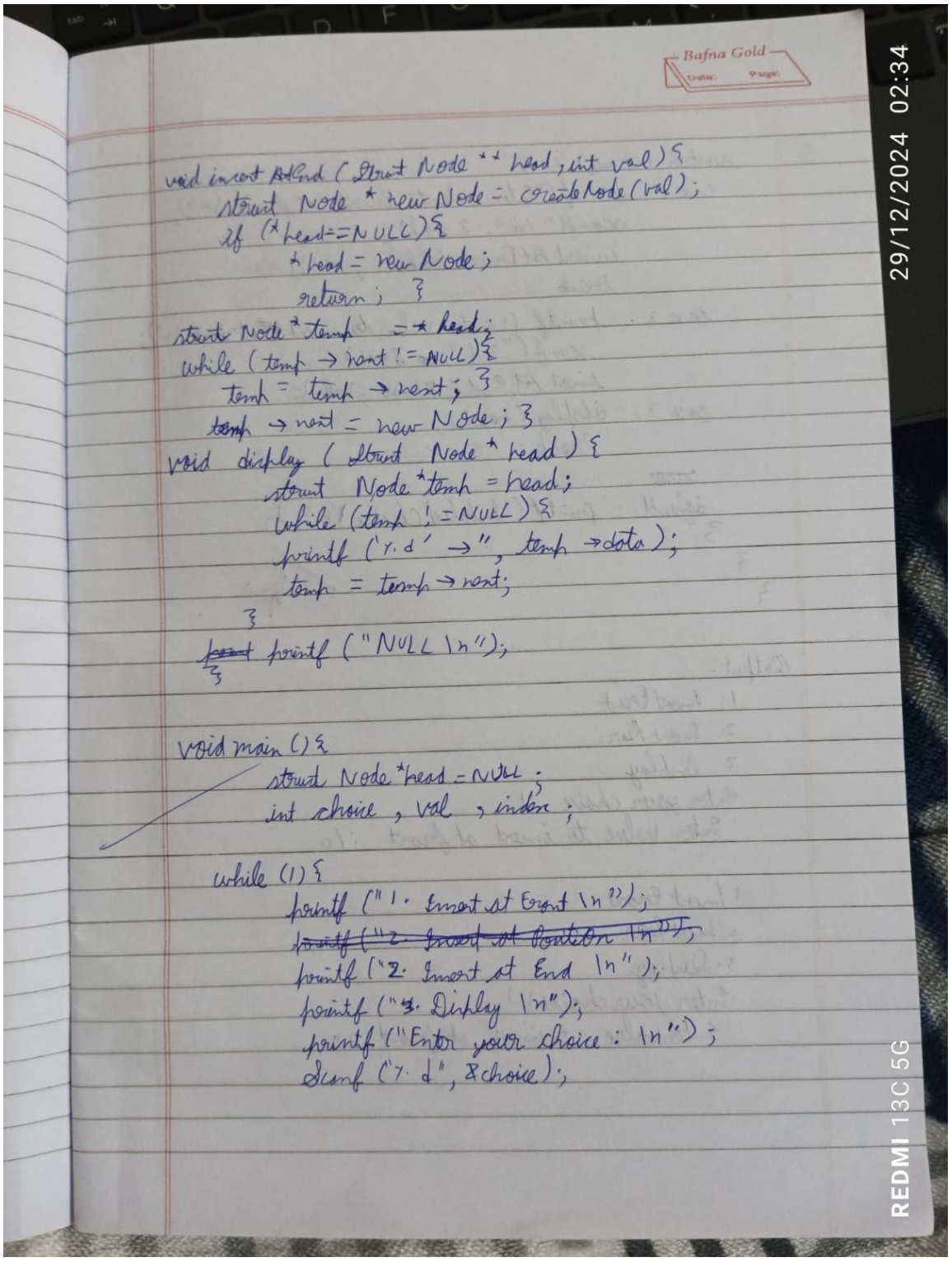
```
    return val;
}

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");
        }
    }
}
```

```
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 1
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 2
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 3
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 4
Overflow
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Popped = 1
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Popped = 2
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Popped = 3
Popped = 3
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 2
Underflow
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 5
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 1
Enter the value to insert: 7
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 3
5 7
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 8
Invalid choice
1 for insert
2 for delete
3 for display
4 for exit
Enter your choice: 4
Exiting...

Process returned 0 (0x0)   execution time : 118.059 s
Press any key to continue.
```

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.





```
void insertAtEnd (struct Node ** head, int val) {
    struct Node * newNode = createNode(val);
    if (*head==NULL){
        *head = newNode;
        return; }
struct Node *temp  = *head;
while (temp -> next != NULL){
    temp = temp -> next; }
temp -> next = newNode; }
void display ( struct Node *head ) {
        struct Node *temp = head;
        while (temp != NULL) {
        printf ("%d" -> ", temp -> data);
        temp = temp -> next;
    }
    printf ("NULL \n");
}

void main () {
        struct Node *head = NULL;
        int choice , val , index;

    while (1) {
        printf ("1. Insert at Front \n");
        printf ("2. Insert at Position \n");
        printf ("2. Insert at End \n");
        printf ("3. Display \n");
        printf ("Enter your choice : \n");
        scanf ("%d", &choice);
```

```
switch (choice) {
  case 1: printf("Enter value to insert at front: \n");
          scanf("%d", &val);
          insertAtFront(&head, val);
          break;
  case 2: printf("enter value to insert at end: \n");
          scanf("%d", &val);
          insertAtEnd(&head, val);
  case 3: display(head);
          break;

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

Output:

1. Insert Front
2. Insert Rear
3. Display

Enter your choice :1

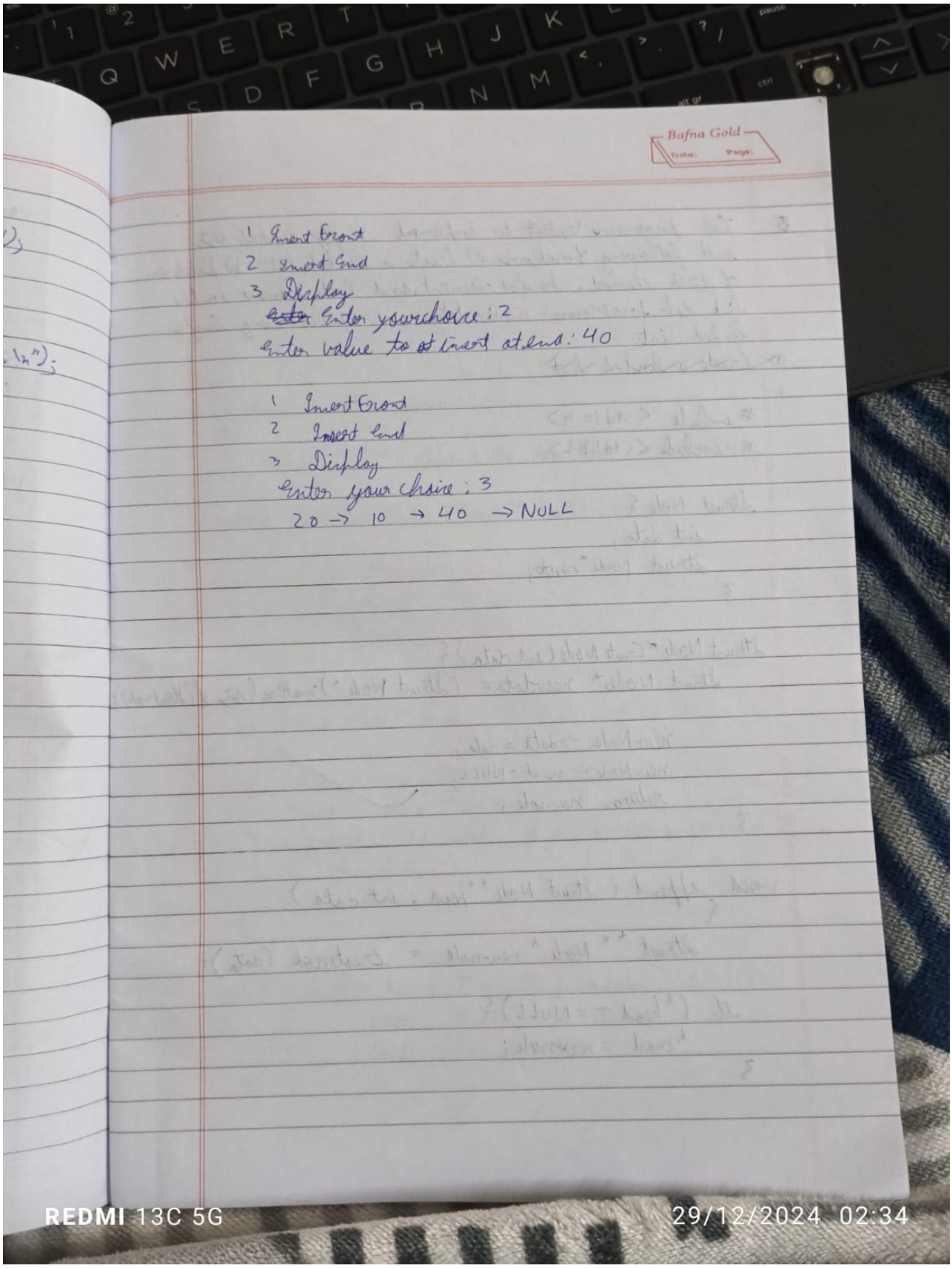
Enter value to insert at front :10

1 Insert Front
2. Insert Rear
3. Display

Enter your choice: 1

Enter value to insert at front: 20

REDMI 13C 5G 29/12/2024 02:34


Bafna Gold
1 Insert Front
2 Insert End
3 Display
Enter your choice: 2
Enter value to be inserted at end: 40
1 Insert Front
2 Insert End
3 Display
Enter your choice: 3
20 → 10 → 40 → NULL
REDMI 13C 5G
29/12/2024 02:34

Code:

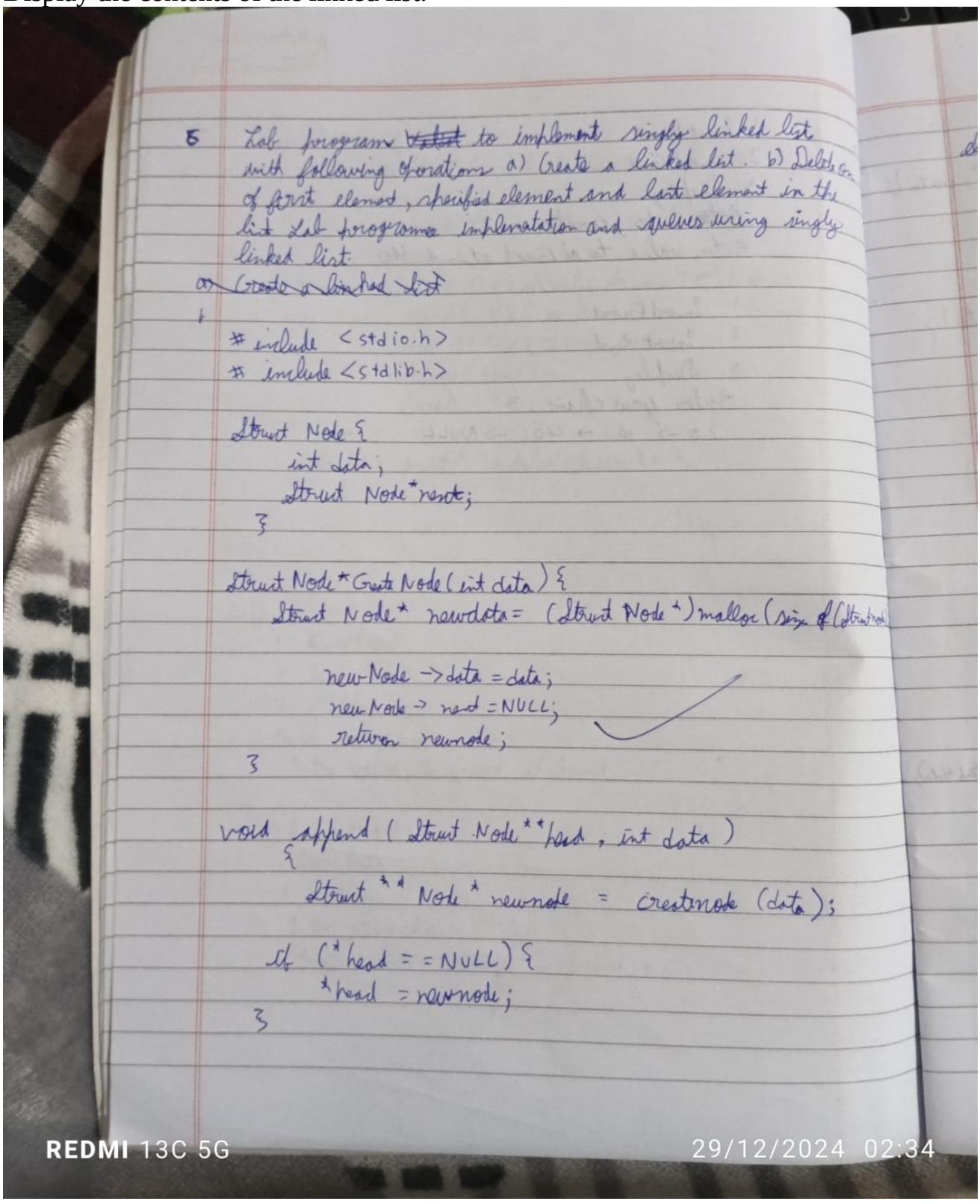
```
#include <stdio.h>
#include <stdlib.h>

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;
}
```

```
Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 2
Enter a value to insert at the beginning: 1

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 2
Enter a value to insert at the beginning: 1

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 3
Enter a value to insert at the end: 3

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 1
1 -> 1 -> 10 -> 20 -> 30 -> 3 -> NULL

Choose an operation:
1. Display linked list
2. Insert at beginning
3. Insert at end
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 41.926 s
Press any key to continue.
```

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.



5 Lab program ~~to~~ 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 Lab programme implementation and queues using singly linked list.

a) ~~Create a linked list~~

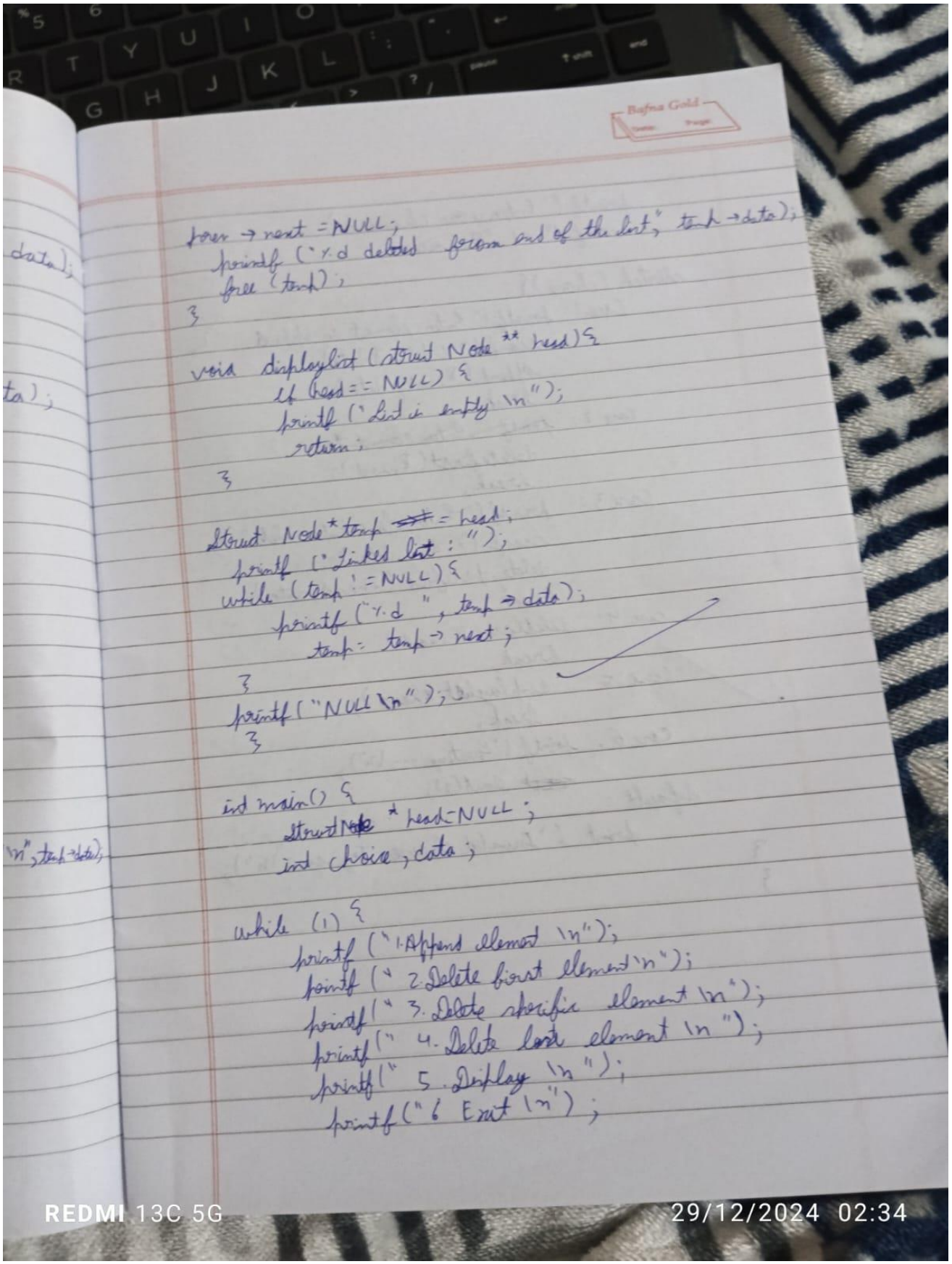
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* CreateNode(int data) {
    struct Node* newdata = (struct Node*)malloc(sizeof(struct
    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;
}
```

REDMI 13C 5G 29/12/2024 02:34

```
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", data
        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 of list \n", temp->
    free (temp)
    head = NULL
    return;
}
while (temp -> next != NULL) {
        prev = temp;
        temp = temp -> next;
    }
```

REDMI 13C 5G 29/12/2024 02:34



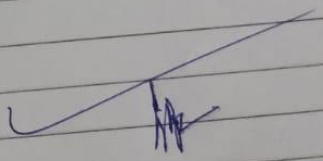
```
    prev->next = NULL;
    printf("%d deleted from end of the list", temp->data);
    free(temp);
}

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("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\n");
    printf("6 Exit\n");
```

REDMI 13C 5G 29/12/2024 02:34

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

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

Enter your choice 15

14 → 15 → NULL

REDMI 13C 5G 29/12/2024 02:34

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

switch(choice){
    case 1: printf("Enter element to append: ");
            scanf("%d", &data);
            append(&head, data);
            break;
    case 2: ~~printf("Enter element to~~
            deletefirst(&head);
            break;
    case 3: printf("Enter element to delete:");
            scanf("%d", data);
            delete specified(&head, data);
            break;
    case 4: deletlast(&head);
            break;
    case 5: displaylist(&head);
            break;
    case 6: printf("Exiting ---\n");
            ~~exit~~ exit(0);
    default:
            print("Invalid choice. Try again\n");
}
}
```

REDMI 13C 5G 29/12/2024 02:34

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;
}
```

```
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 1
Enter the value to insert: 1
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 1
Enter the value to insert: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 5
Element: 1
Element: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 5
Element: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 1
Enter the value to insert: 6
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 4
Enter the value to delete: 6
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 5
Element: 2
1: Insert at end
2: Delete at first
3: Delete at last
4: Delete by value
5: Display
6: Exit
Enter your choice: 6
Exiting...

Process returned 0 (0x0)   execution time : 88.490 s
Press any key to continue.
```

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");
```

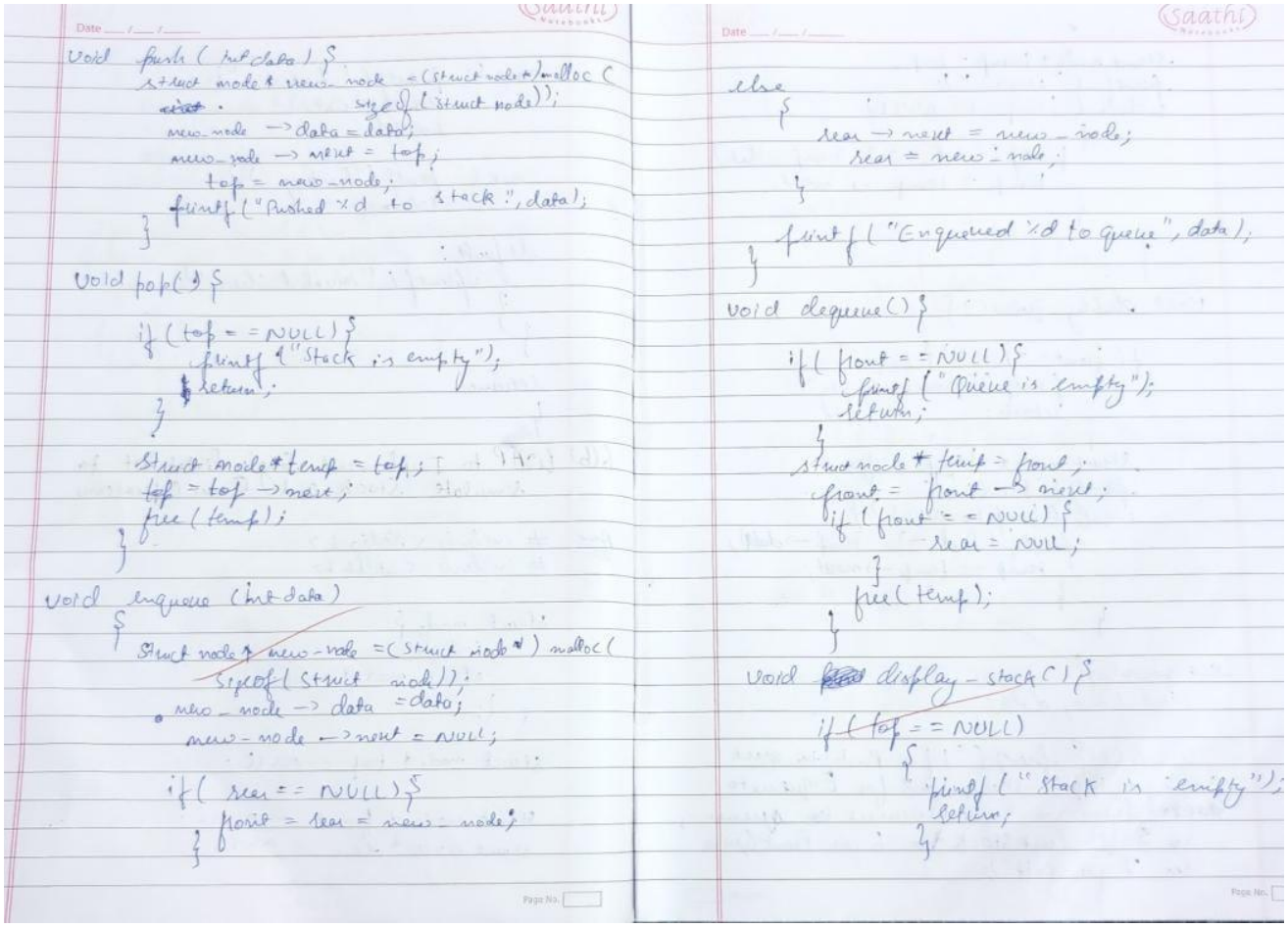
```
printf("6 for reverse list1\n");
printf("7 for reverse list2\n");
printf("8 for exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

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;
        }
    }
}
```

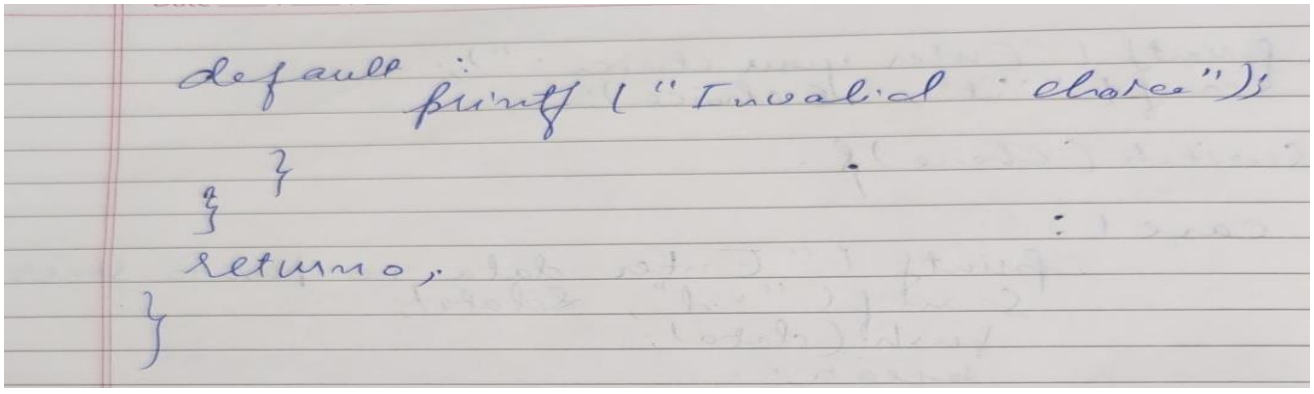
```
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 1
Add to list1: 1
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 1
Add to list1: 2
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 2
Add to list2: 3
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 2
Add to list2: 34
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
```

```
Enter your choice: 7
Reversing list2...
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 4
List1: 2 -> 1 -> NULL
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 5
List2: 34 -> 3 -> NULL
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 3
1 for append to list1
2 for append to list2
3 for concatenation
4 for display list1
5 for display list2
6 for reverse list1
7 for reverse list2
8 for exit
Enter your choice: 4
List1: 2 -> 1 -> 34 -> 3 -> NULL
```

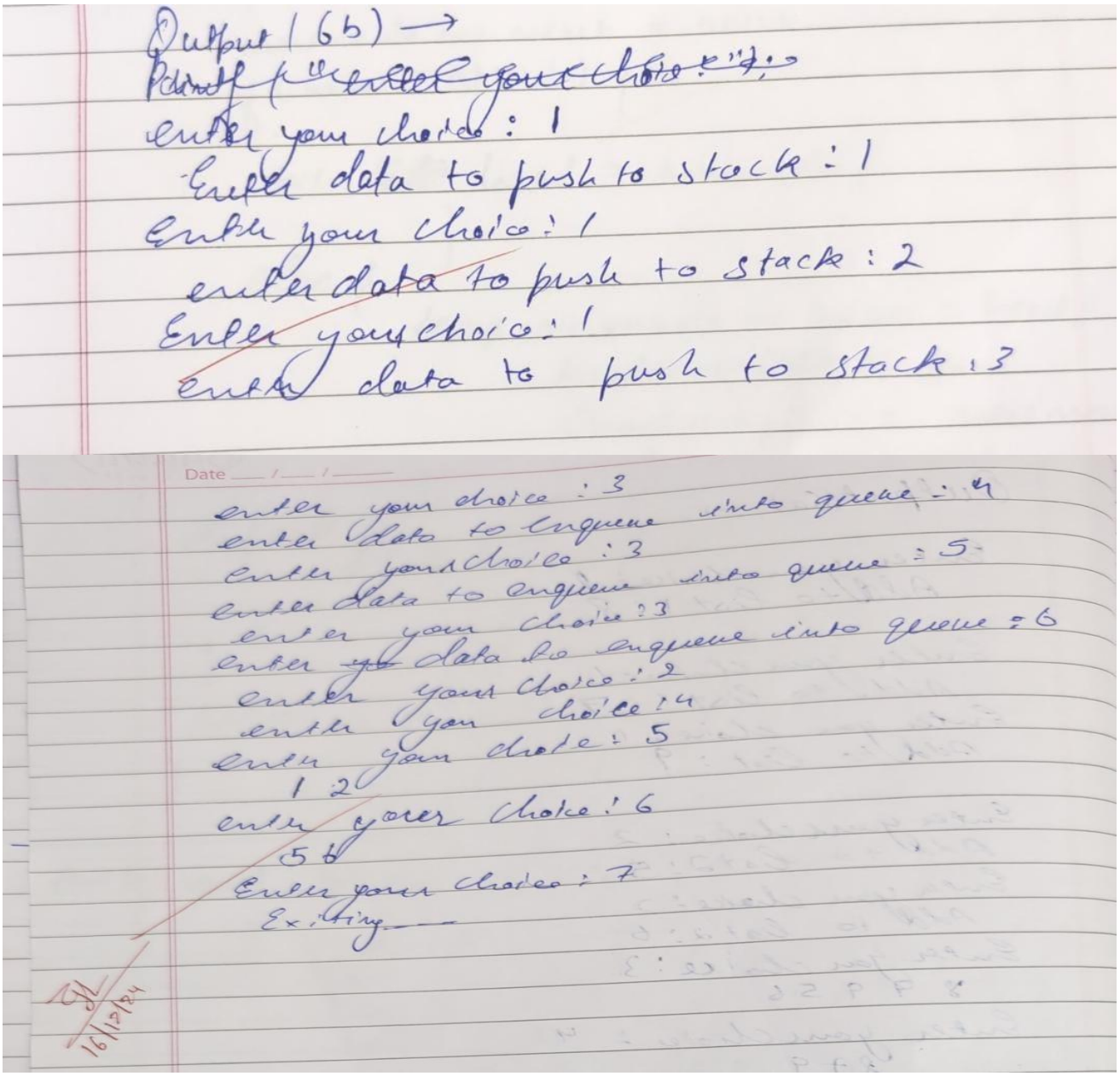
6) b) WAP to Implement Single Link List to simulate Stack & Queue Operations.



```
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", data);
}

void pop() {
    if (top == NULL) {
        printf ("Stack is empty");
        return;
    }
    struct node * temp = top;
    top = top -> next;
    free (temp);
}

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", data);
}

void dequeue() {
    if (front == NULL) {
        printf ("Queue is empty");
        return;
    }
    struct node * temp = front;
    front = front -> next;
    if (front == NULL) {
        rear = NULL;
    }
    free (temp);
}

void display_stack() {
    if (top == NULL)
    {
        printf ("stack is empty");
        return;
    }
```



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



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;
```

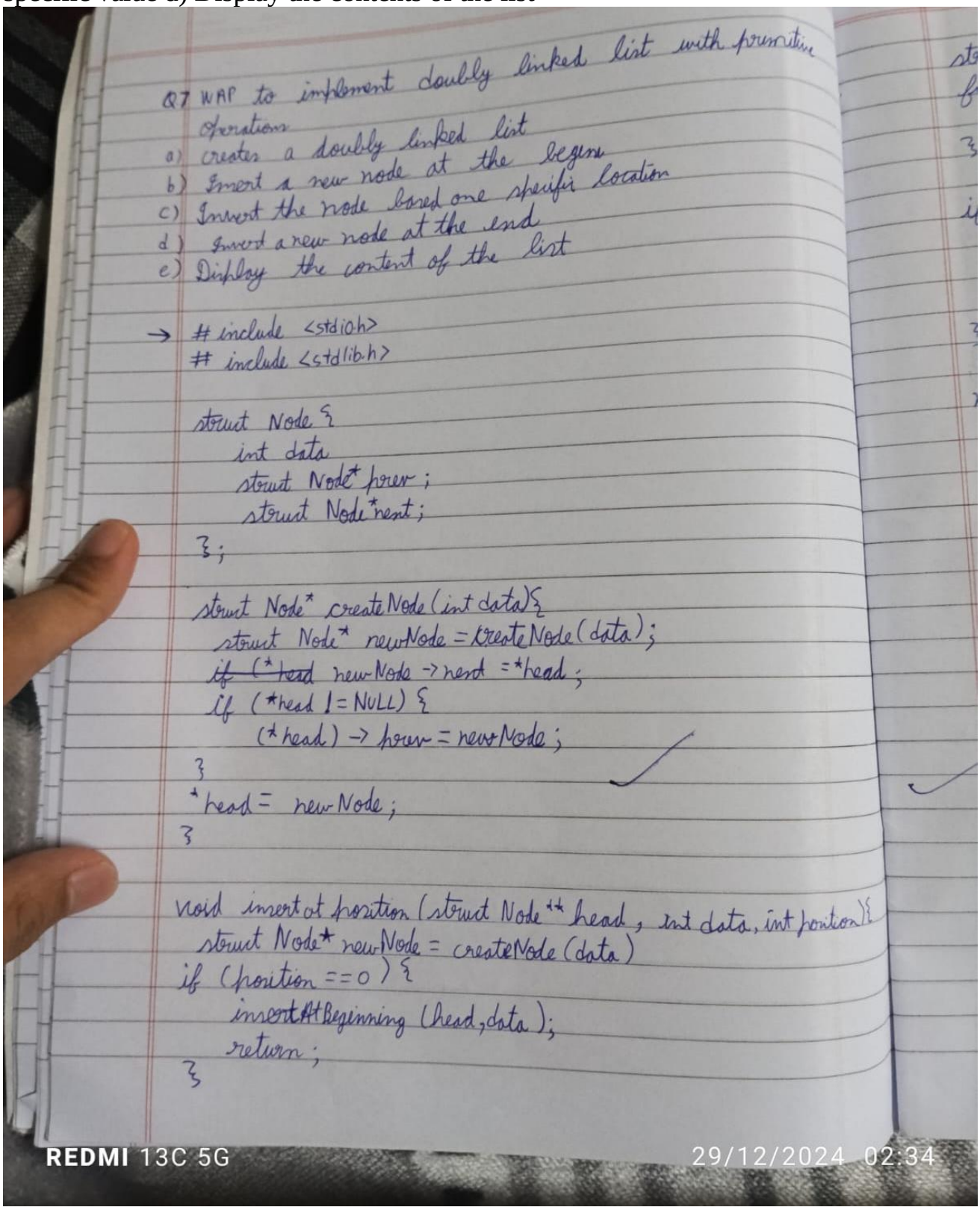
```
while (1) {
    printf("\n1. Push to Stack\n");
    printf("2. Pop from Stack\n");
    printf("3. Display Stack\n");
    printf("4. Enqueue to Queue\n");
    printf("5. Dequeue from Queue\n");
    printf("6. Display Queue\n");
    printf("7. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &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;
}
```

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

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

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

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 2
Popped 3 from stack

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 3
Stack: 2 -> 1 -> NULL
```

```
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 4
Enter data to enqueue to queue: 7
Enqueued 7 to queue

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

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 5
Dequeued 7 from queue

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 9 -> NULL

1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice:
```

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



Q7 WAP to implement doubly linked list with primitive operations
a) creates a doubly linked list
b) Insert a new node at the begin
c) Invert the node based one specific location
d) Invert a new node at the end
e) Display the content of the list

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

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

struct Node* createNode(int data){
struct Node* newNode = createNode(data);
~~if (*head~~ newNode->next = *head;
if (*head != NULL) {
(*head)->prev = newNode;
}
*head = newNode;
}

void insertatPosition(struct Node **head, int data, int position){
struct Node* newNode = createNode(data)
if (position == 0) {
insertAtBeginning(head, data);
return;
}

REDMI 13C 5G 29/12/2024 02:34

```
struct Node* current = *head;
for (int i=0 ; i < position -1 && current != NULL ; i++) {
    current = current -> next;
}

if (current == NULL) {
    printf ("Position out of bounds \n");
    free (newNode);
    return ;
}

newNode -> next = current -> node;
newNode -> prev = current;
current -> next = newNode;

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

void insertAtEnd ( struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }

struct Node* current = head;
while (current -> next != NULL) {
    current = current -> next
}
    current -> next = newNode;
    newNode -> next current;
}
printf ("\n");  }
```

REDMI 13C 5G 29/12/2024 02:35

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

  insertAtEnd (&head, 10);
  insertAtEnd (&head, 20);
  insertAtEnd (&head, 30);
  printf ("List after intrerting at the end:");
  displayList (head);

  insertAtBeginning (&head, 5);
  printf ("List after inserting at the beginning:");
  display List (head);

  insertAtPosition (&head, 15, 2);
  printf ("List after inserting 15 at position 2:");
  display List (head);
  insertAtEnd (&head, 40);
  printf ("List after inserting 40 at the end:");
  display List (head);

  return 0;
}
```

OUTPUT:

List after insertion at end: 10 20 30
List after insertion at beginning: 5 10 20 30
List after insertion at position 2: 5 10 15 20 30

Q.8 Write
a) To
b) To
prec
#in
#i
str
st

REDMI 13C 5G 29/12/2024 02:35

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;
```

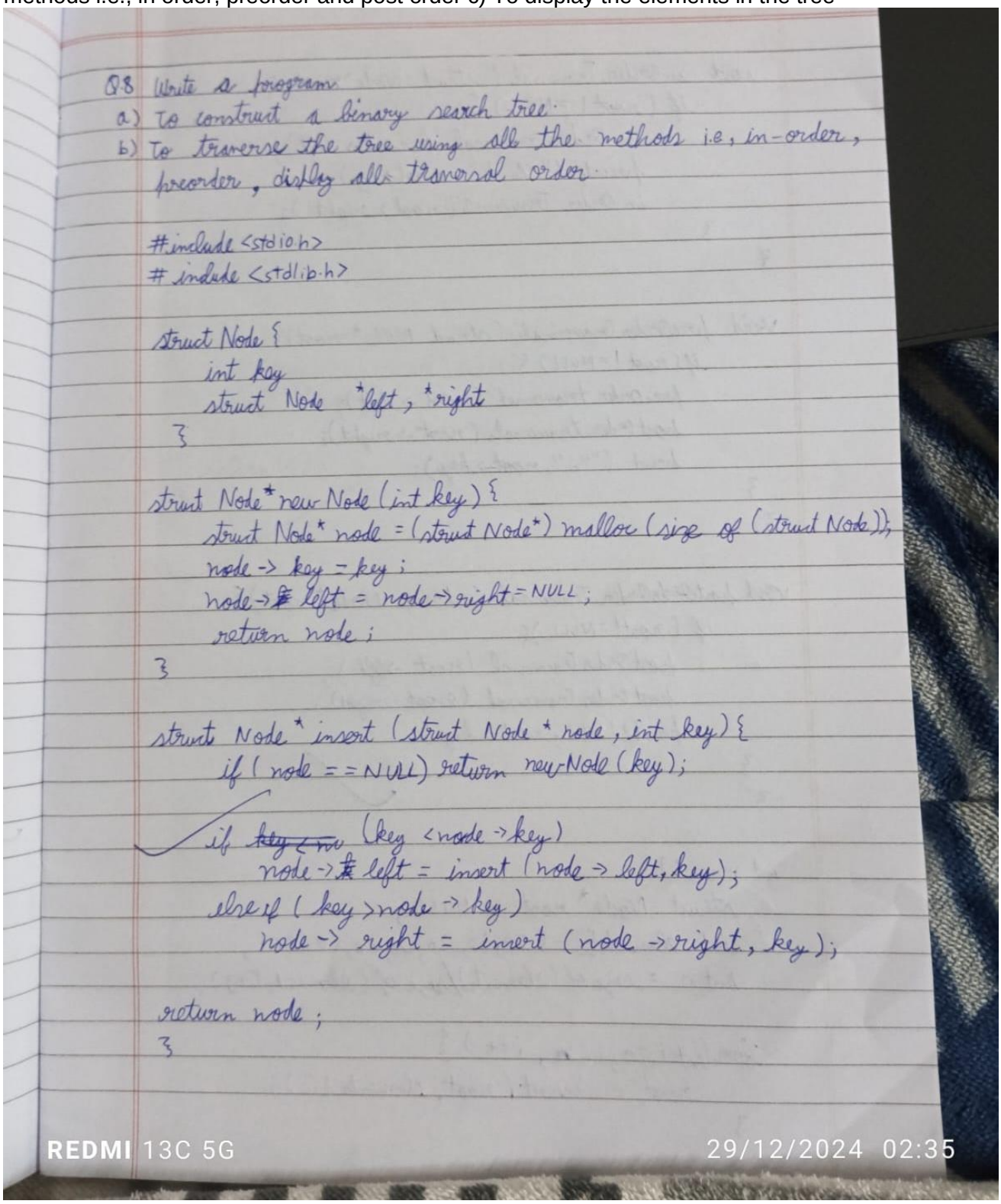
```
    }
}

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;
}
```

```
Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 1
Enter value to add at the beginning: 22

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 1
Enter value to add at the beginning: 45

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 2
Enter value to add at the end: 78

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 3
Enter value to insert at a position: 1
Enter the position: 1

Doubly Linked List Menu:
1. Add at the beginning
2. Add at the end
3. Insert at a position
4. Print the list
5. Exit
Enter your choice: 4
1 <-> 45 <-> 22 <-> 78 <-> NULL
```

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



Q.8 Write a program
a) To construct a binary search tree.
b) To traverse the tree using all the methods i.e, in-order, preorder, display all traversal order.

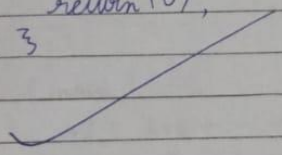
```
#include <stdio.h>
#include <stdlib.h>

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;
}
```

REDMI 13C 5G 29/12/2024 02:35

```
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) {
        preorder traversal (root->left);
        postorder traversal (root->right);
        print ("%d", root->key);
    }
}

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

int main (){
    struct Node * root = NULL;
    int elements[] = {50, 30, 20, 40, 70, 60, 80};
    int n = size of (elements)/size of (elements[0]);

    for(int i=0; i<n; i++) {
        root = insert (root, elements[i]);
    }
```

REDMI 13C 5G 29/12/2024 02:35

```
printf("In-order traversal :");
inOrderTraversal(root);
print ("\n");

printf("Pre-Order Traversal :");
preOrderTraversal(root);
printf("\n");

printf("Post-order traversal:");
postOrderTraversal(root)
printf("\n");

return(0);
}
```



REDMI 13C 5G 29/12/2024 02:35

Code:

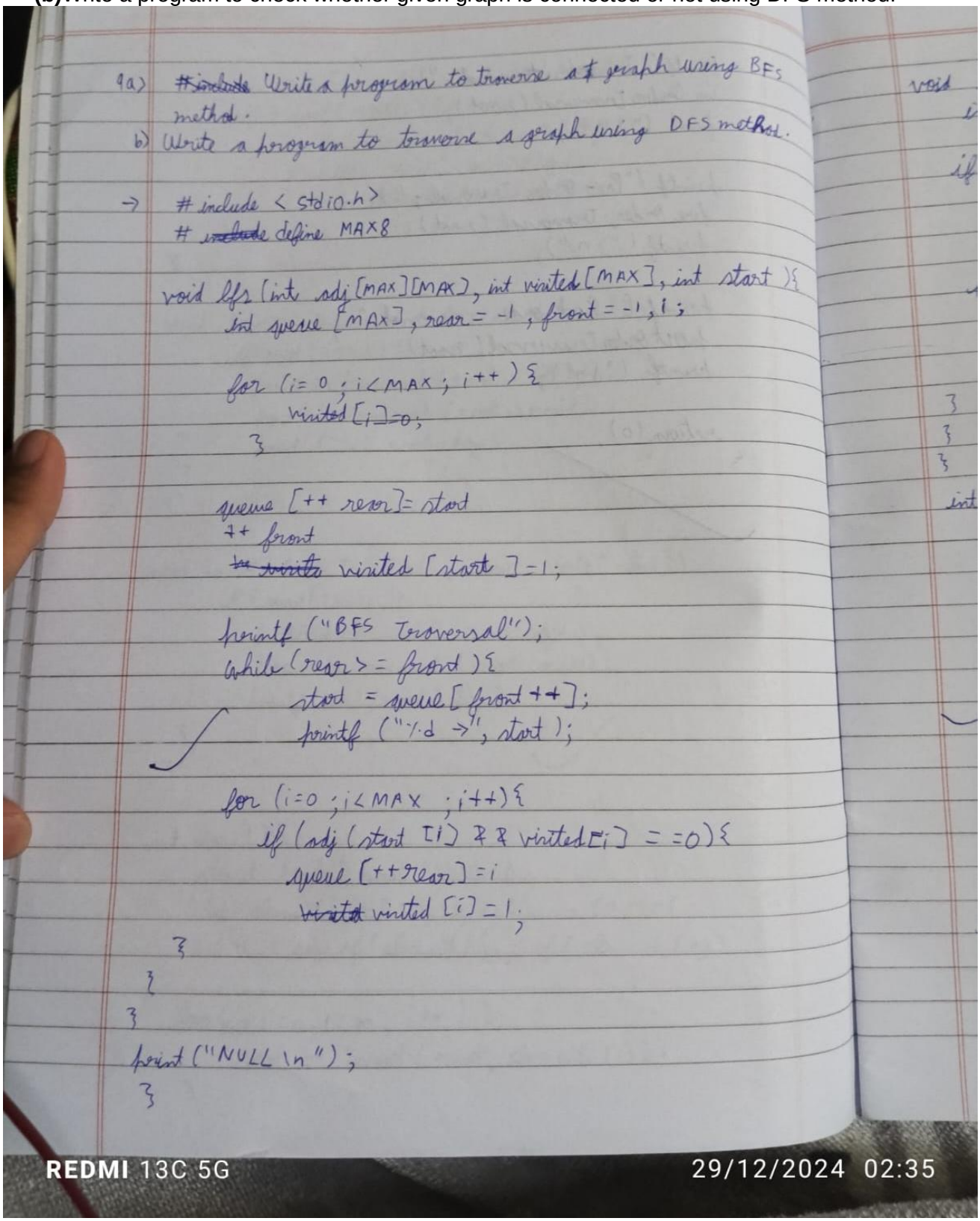
```
#include <stdio.h>
#include <stdlib.h>

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;
}
```

```
In-order Traversal: 20 30 40 50 60 70 80
Pre-order Traversal: 50 30 20 40 70 60 80
Post-order Traversal: 20 40 30 60 80 70 50

Process returned 0 (0x0)   execution time : 0.005 s
Press any key to continue.
```

10(a) Write a program to traverse a graph using BFS method.

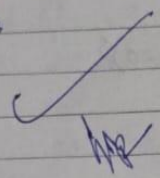
(b) Write a program to check whether given graph is connected or not using DFS method.



```
void dfs (int adj[MAX][MAX], int visited [MAX], int start){
  int i;

  if (visited[start]==0){
    printf ("%d ->", start);
    visited [start]=1;

  for (i=0; i<MAX ; i++){
    if (adj [start [i] && visited [i]==0){
      dfs (adj, visited, i);
    }
  }
}
}

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 };

  }

print ("BFS Traversal \n");
  bfs (adj, visited, 0 );
```

REDMI 13C 5G 29/12/2024 02:35

```
for (int i=0; i<MAX ; i++ ){
        visited [i] = 0;
}
printf ("\n DFS Traversal \n");
  dfs (adj, visited , 0);

result 0;
}
```



REDMI 13C 5G 29/12/2024 02:35

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;
}
```

```
BFS Traversal
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> NULL

Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```

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;
}
```

```
DFS Traversal: 0 -> 2 -> 3 -> 4 -> 1 -> NULL

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```