VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

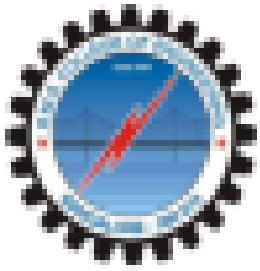
Data Structures using C Lab

(23CS3PCDST)

Submitted by

Umesha H N (1BM24CS428)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Data Structures using C Lab (23CS3PCDST)"carried out by **Umesha H N(1BM24CS428),** who is bonafide student of **B.M.S.College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. 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.

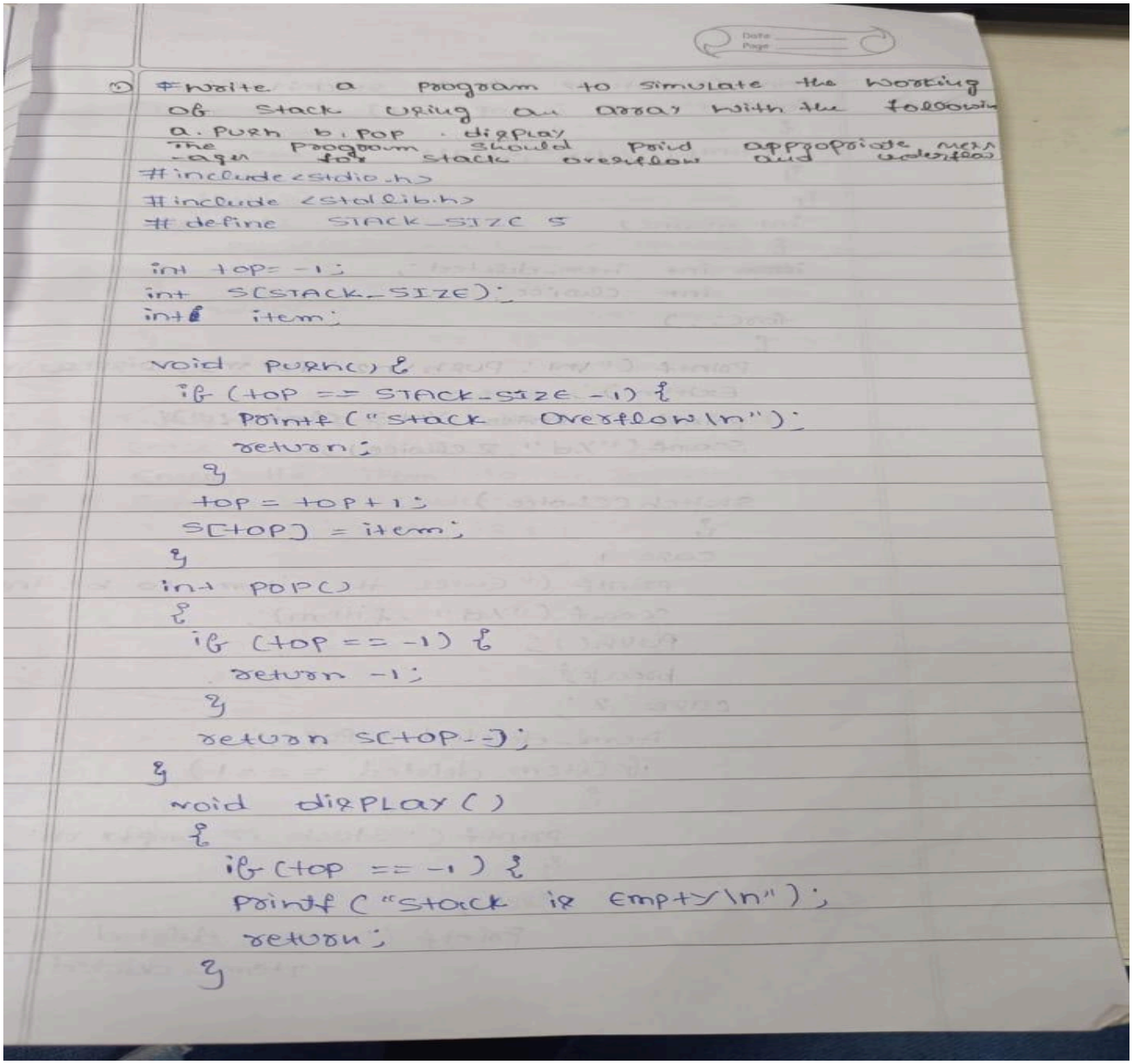
Lab faculty Incharge Name Ramya K M Assistant Professor **Department of CSE, BMSCE**	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link :https://github.com/Umeshahnn/1BM24CS428_DSA

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



```
#Write a program to simulate the working
of stack using an array with the following
a. Push b. Pop . display
The program should print appropriate mess
-ages for stack overflow and underflow
#include<stdio.h>
#include <stdlib.h>
# define STACK_SIZE 5

int top= -1;
int S[STACK_SIZE];
int item;

void PUSH() {
  if (top == STACK_SIZE -1) {
    Printf ("Stack Overflow\n");
    return;
  }
  top = top+1;
  S[top] = item;
}
int POP()
{
  if (top == -1) {
    return -1;
  }
  return S[top--];
}
void display()
{
  if (top == -1) {
  Printf ("Stack is Empty\n");
  return;
}
```

```
Printf("contents of the stack:\n");
for(int i=0; i<=top; i++)
 {
  Printf("%d\n",s[i]);
 }
}
int main()
 {
  int item_deleted;
      int choice;
 for(;;)
  {
   Printf ("\n1: Push\n2: Pop\n3: Display\n4:
   Exit\n");
   Printf ("Enter your choice:");
   Scanf ("%d", &choice);

  switch(choice)
   {
   case 1:
     Printf ("Enter the item to be inserted:");
     Scanf ("%d", &item);
     Push();
     break;
  case 2:
     item_deleted = Pop();
     if (item_deleted == -1)
       {
         Printf ("Stack is empty\n");
       }
       else {
        Printf ("Item deleted is %d\n",
                 Item_deleted); }
```

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

Seen

output

```
1. PUSH  2. POP  3: display  4. Exit
Enter your choice: 1
Enter the item to be inserted: 23
Enter your choice: 2
Item deleteded is 23
Enter your choice: 3
23, 23
Enter your choice: 3
Stack is empty
Enter your choice: 2
Stack Underflow, Stack is empty
Enter your choice: 1
1, 1, 1, 1, 1
Enter your choice: 1
Stack is overflow
```

Code :

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

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

void push(int value) {
   if (top == MAX - 1) {
      printf("Stack Overflow! Cannot push %d\n", value);
   } else {
      top++;
      stack[top] = value;
      printf("Pushed %d onto the stack.\n", value);
   }
}

void pop() {
   if (top == -1) {
      printf("Stack Underflow! Cannot pop from an empty stack.\n");
   } else {
      printf("Popped %d from the stack.\n", stack[top]);
      top--;
   }
}

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

int main() {
   int choice, value;

   while (1) {
      printf("\nStack Operations:\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Display\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
switch (choice) {
         case 1:
            printf("Enter the value to push: ");
            scanf("%d", &value);
```

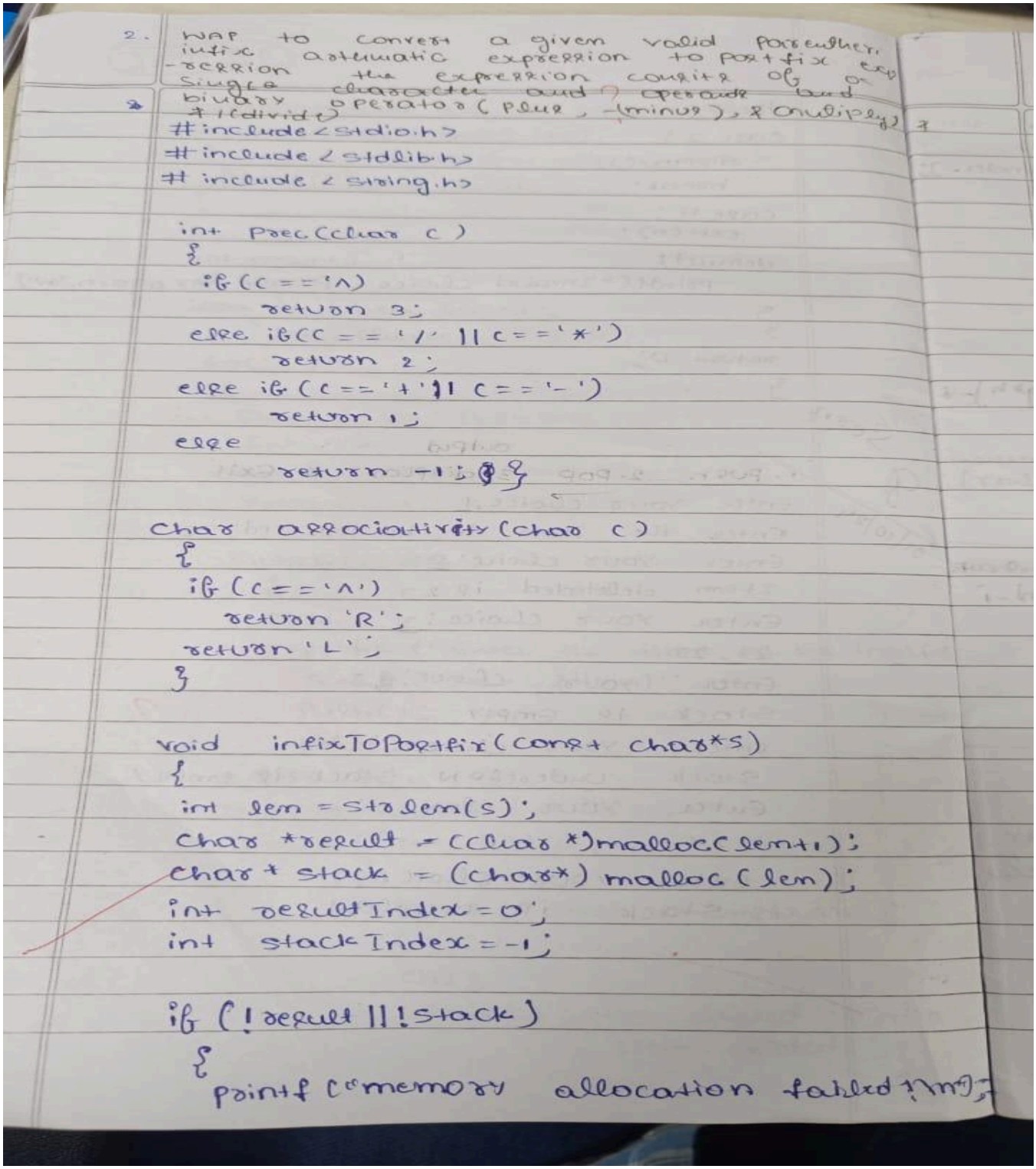
```
            push(value);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting program.\n");
            return 0;
        default:
            printf("Invalid choice! Please enter a number between 1 and 4.\n");
    }
  }
}
```

OUTPUT :

```
Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 25
Pushed 25 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 10
Pushed 10 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped 10 from the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 15
Pushed 15 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements: 25 15

Stack Operations:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
Exiting program.
PS C:\Users\Admin\Desktop\1BM23CS314\C>
```

Program 2:

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



```
2. WAP to convert a given valid parenthesized infix arithmetic expression to postfix exp
-ression the expression consists of single character and operands and
binary operators (plus, -(minus), * (multiply)
#/(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]
                    //remaining !='(')
  {
   result[resultIndex++]=stack[stackIndex--];
  }
  stackIndex--;
 }
else {
  while(stackIndex >=0 && (prec(c)<prec(stack[
        stackIndex]) || (prec(c)==prec(stack[stac
        Index]) == '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;
 }
```

```
scanf
char exp[];
printf("enter expression");
scanf("%s", &exp);
```

Seen.

O/P

abcd^e-fgh*+^*i-

Explanation

Postfix syntax: <operand> <operand> <operator>

a+b*(c^d-e)^(f+g*h)-i

= ab

= a+b*(

O/P = a+b(.

<operand> <operand> <operator> → prefix

<operator> <operand> <operand>

<operand> <operator> <operand> infix

~~top~~

a+b*(c^d-e)^(f+g*h)-i

* a+b*cd^e-^fgh*+-i

* cd^e-fgh*+^

* a+bcd*cd^e^fgh*+^-i

* a+bcd^efgh*+^*-i

* abcd^efgh*+^+-i

* abcd^efgh*+^*+i-;

This is infix to postfix equation.

-2: o||p

The expression of infix to postfix is : abcd^e-fgh*+^*i-

b)

Method 2

a+b*(c^d-e)^(f+g*h)-i

	Stack	Expression
a		a
+	+	a
b		ab
(	+(	ab
*	+(*	ab
c	+(*	abc
^	+(*^	abc
d	+(*^	abcd
-	+(*^-	abcd^
e	+(*-	abcd^e
)	+(*-)	abcd^e
^	+(*-)^	abcd^e
(	+(*-)^(	abcd^e
f	+(*-)^(	abcd^ef
+	+(*-)^(+	abcd^ef
g	+(*-)^(+	abcd^efg
*	+(*-)^(+	abcd^efg*
h	+(*-)^(+	abcd^efg*h
—	+(*-)^(+	abcd^efg*h+^
		abcd^efg*h+^*i-

Code:

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

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




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

void infixToPostfix(const char *s) {
char result[MAX];
char stack[MAX];
int resultIndex = 0;
int stackIndex = -1;
int len = strlen(s);
for (int i = 0; i < len; i++) {
```

```
    char c = s[i];
// If the scanned character is an operand, add it to the output string.
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
            result[resultIndex++] = c;
    } else if (c == '(') {
        // If the scanned character is an '(', push it to the stack.
             stack[++stackIndex] = c;
                                                                                    }
     else if (c == ')') {
// If the scanned character is an ')', pop and add to the output string from the stack until
an '(' is encountered.
        while (stackIndex >= 0 && stack[stackIndex] != '(') {
               result[resultIndex++] = stack[stackIndex--];
                                                                                       }
           stackIndex--; // Pop '('
                                                                                    }
else {
        // If an operator is scanned
        while (stackIndex >= 0 && (prec(c) < prec(stack[stackIndex]) ||
                 (prec(c) == prec(stack[stackIndex]) && associativity(c) == 'L'))) {
               result[resultIndex++] = stack[stackIndex--];
                                                                                    }
           stack[++stackIndex] = c;
                                                                                    }
}


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

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

```
}

int main() {
char exp[MAX];

printf("Enter an infix expression: ");
fgets(exp, MAX, stdin);
exp[strcspn(exp, "\n")] = 0; // Remove trailing newline

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

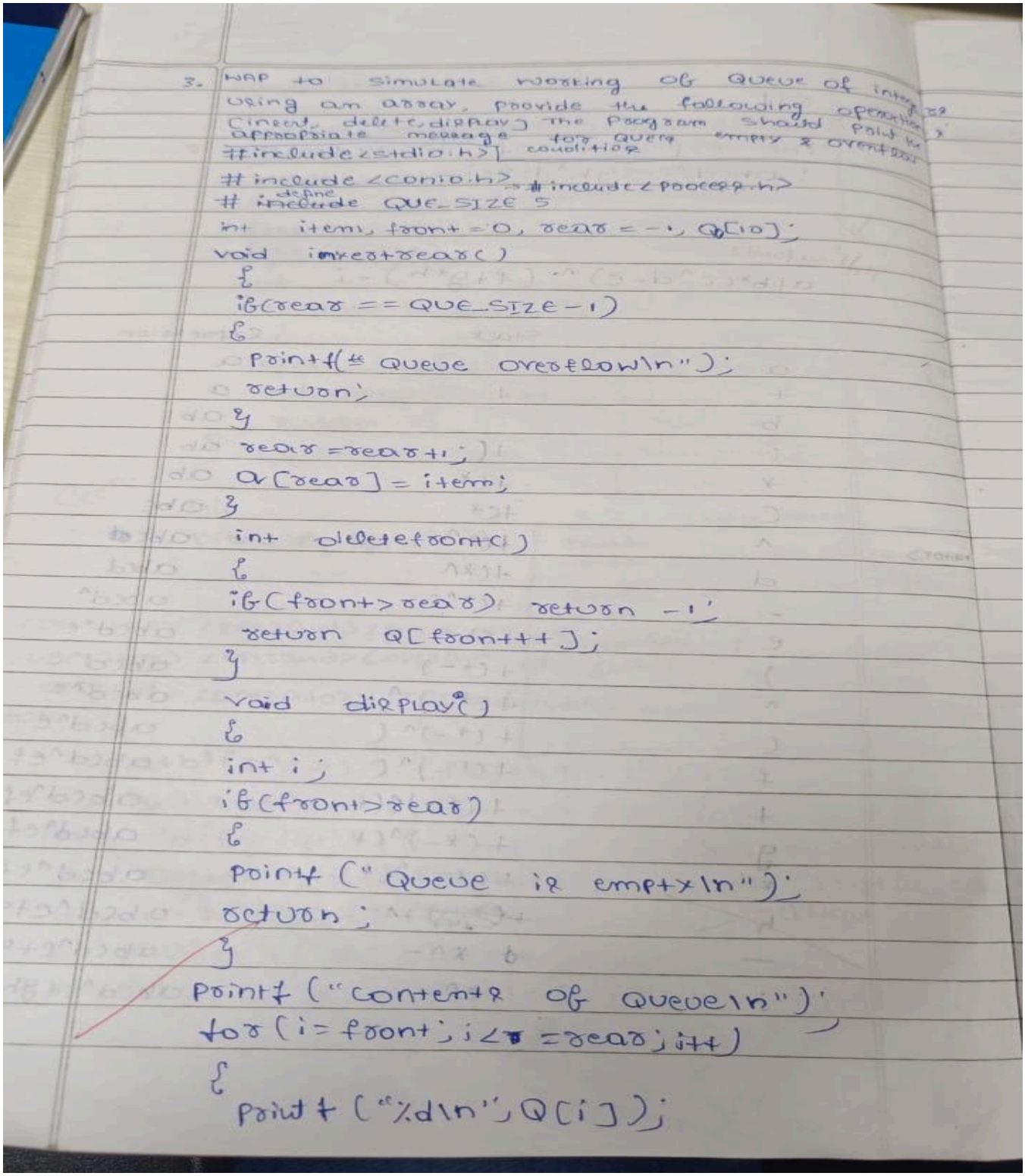
Output:

```
PS C:\Users\Admin\Desktop\1BM23CS314> cd c
PS C:\Users\Admin\Desktop\1BM23CS314\c> gcc .\Queue.c
PS C:\Users\Admin\Desktop\1BM23CS314\c> .\a.exe
Enter an infix expression: a+b*c-q/b+p^g
Postfix expression: abc*+qb/-pg^+
PS C:\Users\Admin\Desktop\1BM23CS314\c>
```

Program 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



```
3. WAP to simulate working of Queue of integers
using an array, provide the following operations
(insert, delete, display) The program should print
appropriate message for queue empty & overflow
conditions
#include<stdio.h>
#include<conio.h>  #include<process.h>
#define QUE_SIZE 5
int item, front = 0, rear = -1, Q[10];
void insert rear()
{
if(rear == QUE_SIZE-1)
{
printf("Queue overflow\n");
return;
}
rear = rear+1;
Q[rear] = item;
}
int deletefront()
{
if(front>rear) return -1;
return Q[front++];
}
void display()
{
int i;
if(front>rear)
{
printf("Queue is empty\n");
return;
}
printf("contents of Queue\n");
for(i=front; i<=rear; i++)
{
printf("%d\n", Q[i]);
```

~~o/p~~

output

1. InsertRear } insert
2. DeleteFront
3. DisRear
4. Exit

Enter the choice:
1
Enter item to be inserted: 11

1. insertRear
2. DeleteFront } delete
3. DisPlay
4. exit

Enter the choice:
2
~~Enter~~ ~~the~~ Item deleted = 11

1. insertRear
2. DeleteFront
3. DisPlay
4. exit

Enter the choice: 1
Enter item to be inserted: 13

1. insertRear
2. DeleteFront
3. DisPlay
4. exit

Enter the choice: 1
Enter item to be inserted: 4

1. Insertrear
2. Deletefront
3. Display
4. Exit

Enter choice: 3

Contents of queue is given below

13, 44

Code:

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

#define MAX 5

// Function to check if the queue is full
int isFull(int rear) {
    if (rear == MAX - 1) {
        return 1;
    }
    return 0;
}

// Function to check if the queue is empty
int isEmpty(int front, int rear) {
    if (front == -1 || front > rear) {
        return 1;
    }
    return 0;
}

// Function to insert an element into the queue
void insert(int queue[], int *front, int *rear, int value) {
    if (isFull(*rear)) {
        printf("Queue Overflow! Cannot insert %d\n", value);
        return;
    }

    if (*front == -1) {
        *front = 0;  // Set front to 0 when inserting the first element
    }

    (*rear)++;
    queue[*rear] = value;
    printf("%d inserted into the queue\n", value);
}



// Function to delete an element from the queue
void delete(int queue[], int *front, int *rear) {
```

```
    if (isEmpty(*front, *rear)) {
        printf("Queue Underflow! No element to delete\n");
        return;
    }

    int deletedValue = queue[*front];
    printf("%d deleted from the queue\n", deletedValue);
    (*front)++;

    // Reset the queue if it becomes empty
    if (*front > *rear) {
        *front = *rear = -1;
    }
}

// Function to display the elements of the queue
void display(int queue[], int front, int rear) {
    if (isEmpty(front, rear)) {
        printf("Queue is empty!\n");
        return;
    }

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




// Main function
int main() {
    int queue[MAX]; // Queue array
    int front = -1, rear = -1; // Initialize front and rear to -1

    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
```

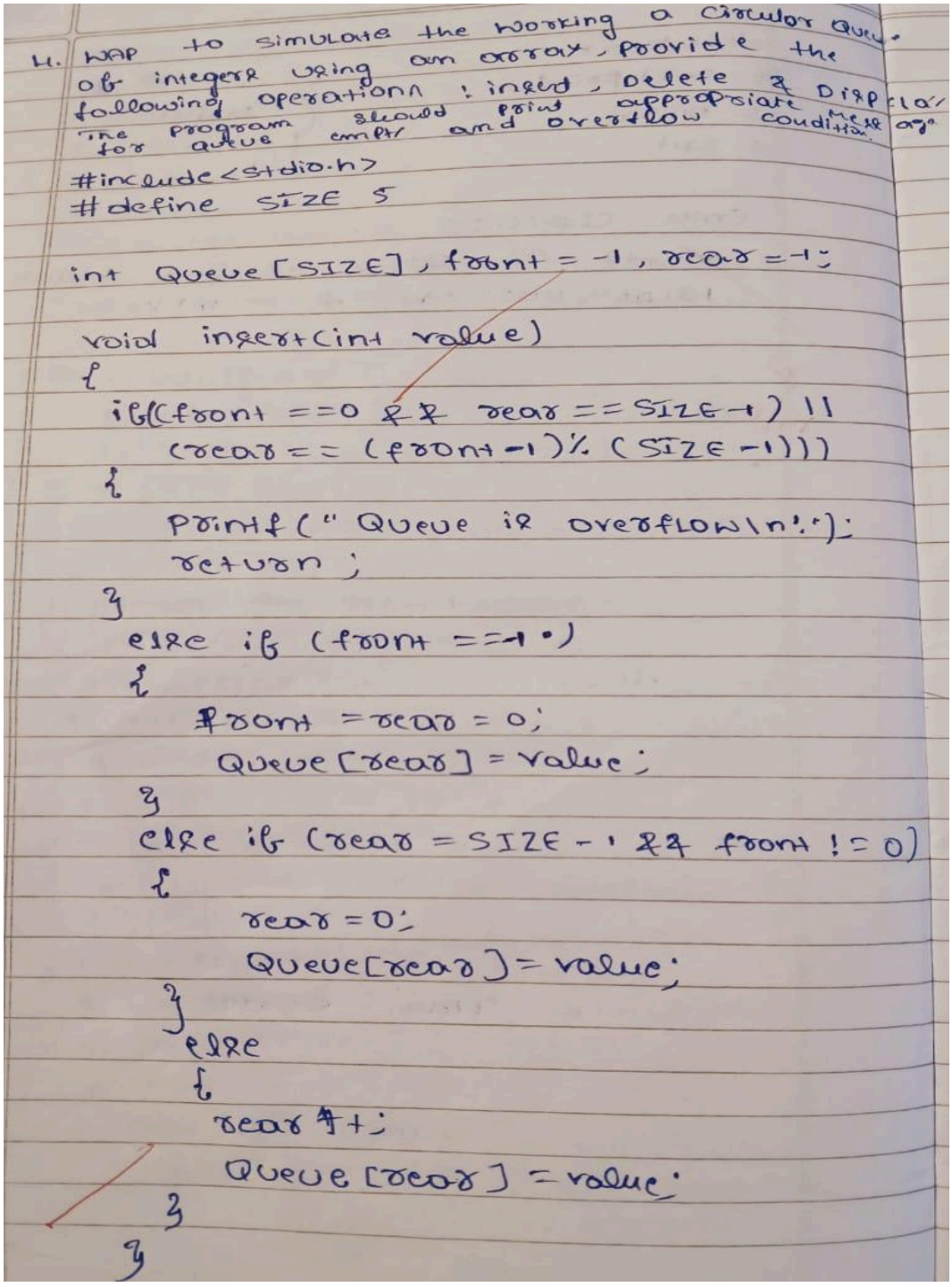
```
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

            case 2:
                delete(queue, &front, &rear);
                break;

            case 3:
                display(queue, front, rear);
                break;

            case 4:
                exit(0);

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

    return 0;
}
```

Output:

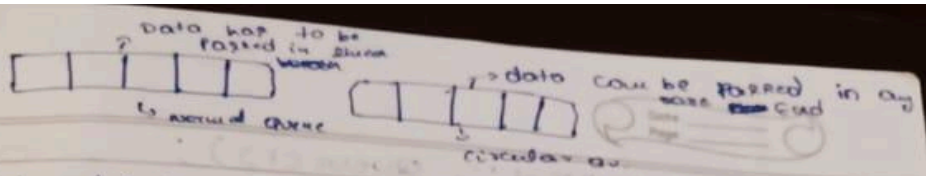
```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 16
16 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 25
25 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
16 deleted from the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 25

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
PS C:\Users\Shashank U\Desktop\C>
```

Program 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



```
4. WAP to simulate the working a circular queue
of integers using an array, provide the
following operations : insert, delete & Display
the program should print appropriate message
for queue empty and overflow condition.

#include<stdio.h>
#define SIZE 5

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

void insert(int value)
{
  if((front ==0 && rear == SIZE-1) ||
     (rear == (front-1)% (SIZE-1)))
  {
     printf(" Queue is overflow\n");
     return ;
  }
  else if (front == -1)
  {
     front = rear = 0;
     Queue[rear] = value;
  }
  else if (rear = SIZE-1 && front != 0)
  {
     rear = 0;
     Queue[rear] = value;
  }
  else
  {
     rear ++;
     Queue[rear] = value;
  }
}
```




```
void delete()
{
 if(front==-1)
 {
   printf("Queue is empty\n");
   return;
 }
   printf("Deleted : %d\n", Queue[front]);
   Queue[front]=-1;

   if (front==rear) {
      front = rear = -1;
    }
   else if(front==SIZE-1)
    {
      front=0;
    }
   else {
      front++;
    }
 }

void display()
{
 if(front==-1) {
 printf("Queue is Empty\n");
 return;
 }
 printf("Queue elements: ");
 if (rear>=front)
 {
 for(int i=front; i<=rear; i++) {
```

```
printf("%d", queue[i]);
 }
}
else {
for(int i= front; i<SIZE; i++)
{
printf("%d", queue[i]);
}
for(int i=0; i<=rear; i++)
 {
  printf("%d", queue[i]);
 }
}
 printf("\n");
}
int main() {
 int choice, value;
 while(1) {
 printf("\n1. Insert\n2. Delete\n3. Dis
-play\n4. Exit\n");
 printf("Enter your choice:");
 scanf("%d", &choice);

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

```
        delete();
        break;

    case 3:
        display();
        break;
    case 4;
        return 0;
    default;
        Printf("Invalid choice, try again
        .);
    }
}
return 0;
}
```

O/P

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5 // Maximum size of the circular queue
// Function to check if the queue is full
int isFull(int front, int rear) {
    if ((rear + 1) % MAX == front) {
        return 1;
    }
    return 0;
}
// Function to check if the queue is empty
int isEmpty(int front, int rear) {
    if (front == -1) {
        return 1;
    }
    return 0;
}
/// Function to insert an element into the queue
void insert(int queue[], int *front, int *rear, int value) {
    if (isFull(*front, *rear)) {
        printf("Queue Overflow! Cannot insert %d\n", value);
        return;
    }
    // If the queue is empty
    if (*front == -1) {
        *front = *rear = 0;
    } else {
        // Circular increment of rear
        *rear = (*rear + 1) % MAX;
    }
    queue[*rear] = value;
    printf("%d inserted into the queue\n", value);
}
// Function to delete an element from the queue
void delete(int queue[], int *front, int *rear) {
    if (isEmpty(*front, *rear)) {
        printf("Queue Underflow! No element to delete\n");
        return;
    }
    int deletedValue = queue[*front];
```

```
    printf("%d deleted from the queue\n", deletedValue);
    // If there is only one element left in the queue
    if (*front == *rear) {
        *front = *rear = -1; // Queue is now empty
    } else {
        // Circular increment of front
        *front = (*front + 1) % MAX;
    }
}
// Function to display the elements of the queue
void display(int queue[], int front, int rear) {
    if (isEmpty(front, rear)) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    if (front <= rear) {
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    } else {
        for (int i = front; i < MAX; i++) {
            printf("%d ", queue[i]);
        }
        for (int i = 0; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}
// Main function
int main() {
    int queue[MAX]; // Queue array
    int front = -1, rear = -1; // Initialize front and rear to -1
    int choice, value;
    while (1) {
        printf("\nCircular Queue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

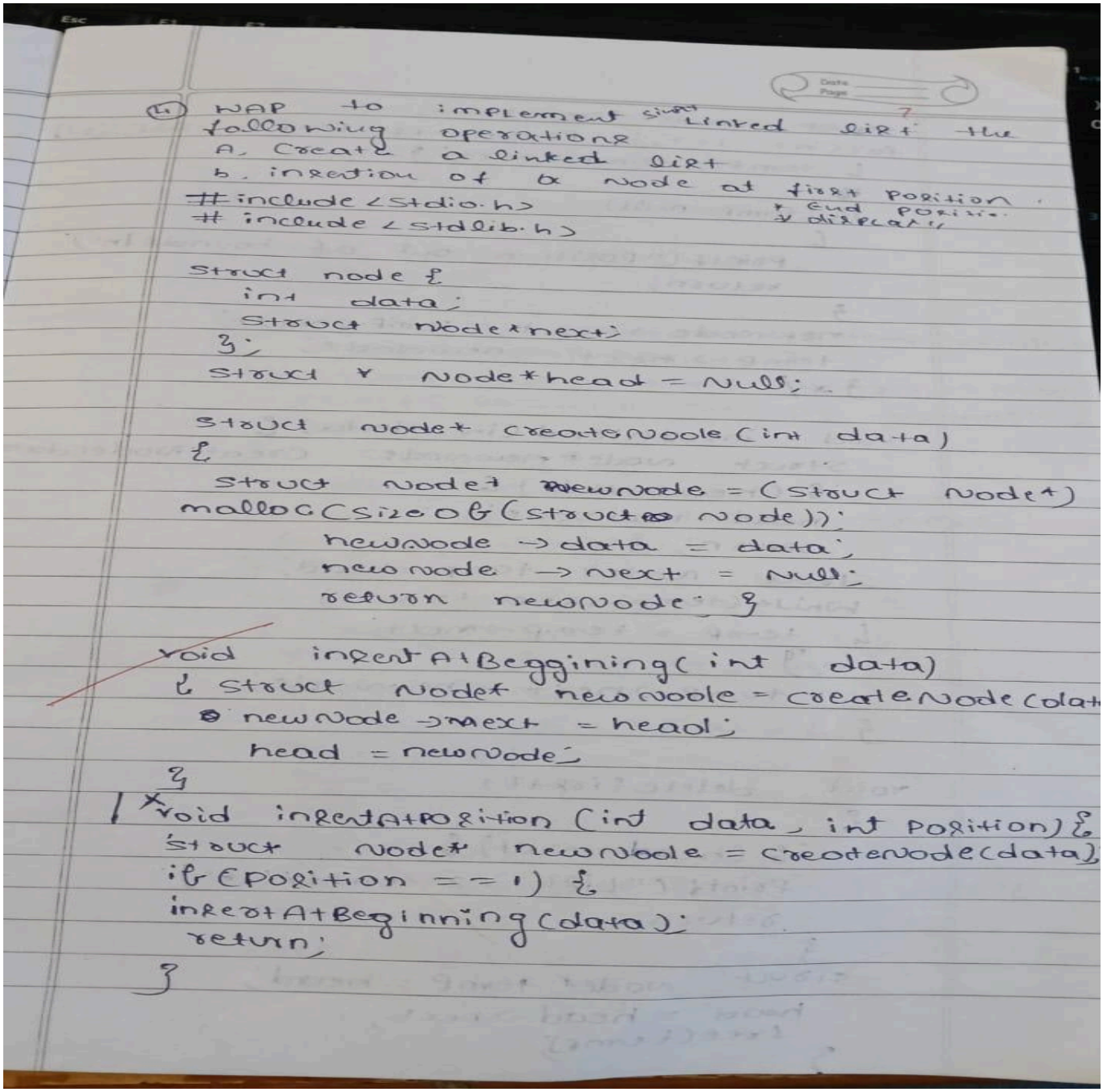
```
        scanf("%d", &choice);

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

Output:

```
Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 12
12 inserted into the queue

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 15
15 inserted into the queue

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
12 deleted from the queue
```

Program 5:

05(a) WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Insertion of a node at **first position** and **at end of list.**
Display the contents of the linked list.
5b) Leetcode problem no.20 (Valid parantheses)



```
struct Node* temp = head;
for(int i = 1; i < position-1 && temp != NULL; i++)
  { temp = temp->next; }
  if (temp = NULL)
    {
      printf("position out of bounds\n");
      return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
  }*/

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

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

```
void displayList()
{
 Struct Node* temp = head;
 if (temp==NULL) {
  Printf ("List is empty\n");
  return; }
While (temp != NULL) {
 Printf("%d ->", temp -> data);
  temp = temp->next;
 }
 Print ("NULL\n");
}

int main()
{
 int choice, data, position;
While (1)
{
  Printf(" Enter 1 to insert the element
     at front\n");
  Printf (" Enter 2 to insert the elem
   -ent at End:");
  Printf (" Enter 3 to Delete the ele
   -ment at front:");
  Printf(" Enter 4 to Delete the elem
   -ent at the end:");
  Printf (" Enter 5 to Display the el
   -ments);
  Printf (" Exit ");

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

```
switch(choice)
{
 case 1:
  Printf(" Enter data : ");
  scanf("%d", &data);
  insert at Beggining (data);
  break;
 case 2:
  Printf (" Enter data : ");
  scanf ("%d", &data);
  insert at ending (data);
  break;
  case 3:
    ~~Delete~~ deletefirst();
        Printf ("deleted list : %d" ~~data~~);
        break;
 case 4:
        deletelast();
        Printf ("deleted list : %d",
        break;
case 5:
        display();
        ~~Printf (" di~~
        break;

case 6:
        exit();
default:
        printf(" Please enter Correct op
}
  return 0;
}
```

Proud

28/10

O/P

Menu:
1. Inserting at beggining
2. Insert at ending
3. Delete at beggining
4. Delete at ending
5. Display the list
6. exit

Enter your choice : 1
Enter data to insert at beggining : 11

Enter your choice : 2
Enter data to insert at Ending : 33

Enter your choice : 5
Linked List : 11 -> 22 ->

Enter your choice : 3
First Node deleted

Enter your choice : 4
Last Node deleted

OR

Enter your choice : 5
Linked list : List is Empty

Enter your choice : 8
Please enter correct choice

Code:

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

// Define the structure for a linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a linked list
void createList(struct Node** head) {
    *head = NULL;
}

// Function to insert a node at the beginning
void insertAtFirst(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
    printf("Node with value %d inserted at the beginning.\n", value);
}

// Function to insert a node at any position
void insertAtPosition(struct Node** head, int value, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (position == 1) {
        newNode->next = *head;
        *head = newNode;
        printf("Node with value %d inserted at position 1.\n", value);
        return;
    }

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

    if (temp == NULL) {
```

```
        printf("Position %d is out of bounds. Insertion failed.\n", position);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
        printf("Node with value %d inserted at position %d.\n", value, position);
    }
}

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        printf("Node with value %d inserted at the end.\n", value);
        return;
    }

    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("Node with value %d inserted at the end.\n", value);
}

// Function to display the contents of the linked list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The 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");
```

```
}

// Main function
int main() {
    struct Node* head;
    createList(&head); // Create an empty list

    int choice, value, position;

    while (1) {
        printf("\nLinked List Operations:\n");
        printf("1. Insert at First\n");
        printf("2. Insert at Position\n");
        printf("3. Insert at End\n");
        printf("4. Display the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtFirst(&head, value);
                break;

            case 2:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                printf("Enter the position to insert at: ");
                scanf("%d", &position);
                insertAtPosition(&head, value, position);
                break;

            case 3:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;

            case 4:
                displayList(head);
```

```
            break;

        case 5:
            exit(0);

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

  return 0;
}
```

Output:

```
Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 1
Enter the value to insert at the beginning: 7
Node with value 7 inserted at the beginning.

Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 2
Enter the value to insert: 8
Enter the position to insert at: 2
Node with value 8 inserted at position 2.

Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 3
Enter the value to insert at the end: 18
Node with value 18 inserted at the end.

Linked List Operations:
1. Insert at First
2. Insert at Position
3. Insert at End
4. Display the list
5. Exit
Enter your choice: 4
Linked List: 7 -> 8 -> 18 -> NULL
```

Leetcode problem:

C Auto

```c
bool isValid(char* s) {
    char stack[strlen(s)];
    int top = -1;
    for (int i = 0; s[i] != '\0'; i++) {
        char current = s[i];
        if (current == '(' || current == '{' || current == '[') {
            stack[++top] = current;
        } else {
            if (top == -1) return false; // Stack is empty, invalid string

            char last = stack[top];
            if ((current == ')' && last == '(') ||
                (current == '}' && last == '{') ||
                (current == ']' && last == '[')) {
                top--; // Pop the stack
            } else {
                return false; // Mismatched bracket
            }
        }
    }
    return top == -1; // Valid if stack is empty
}
```

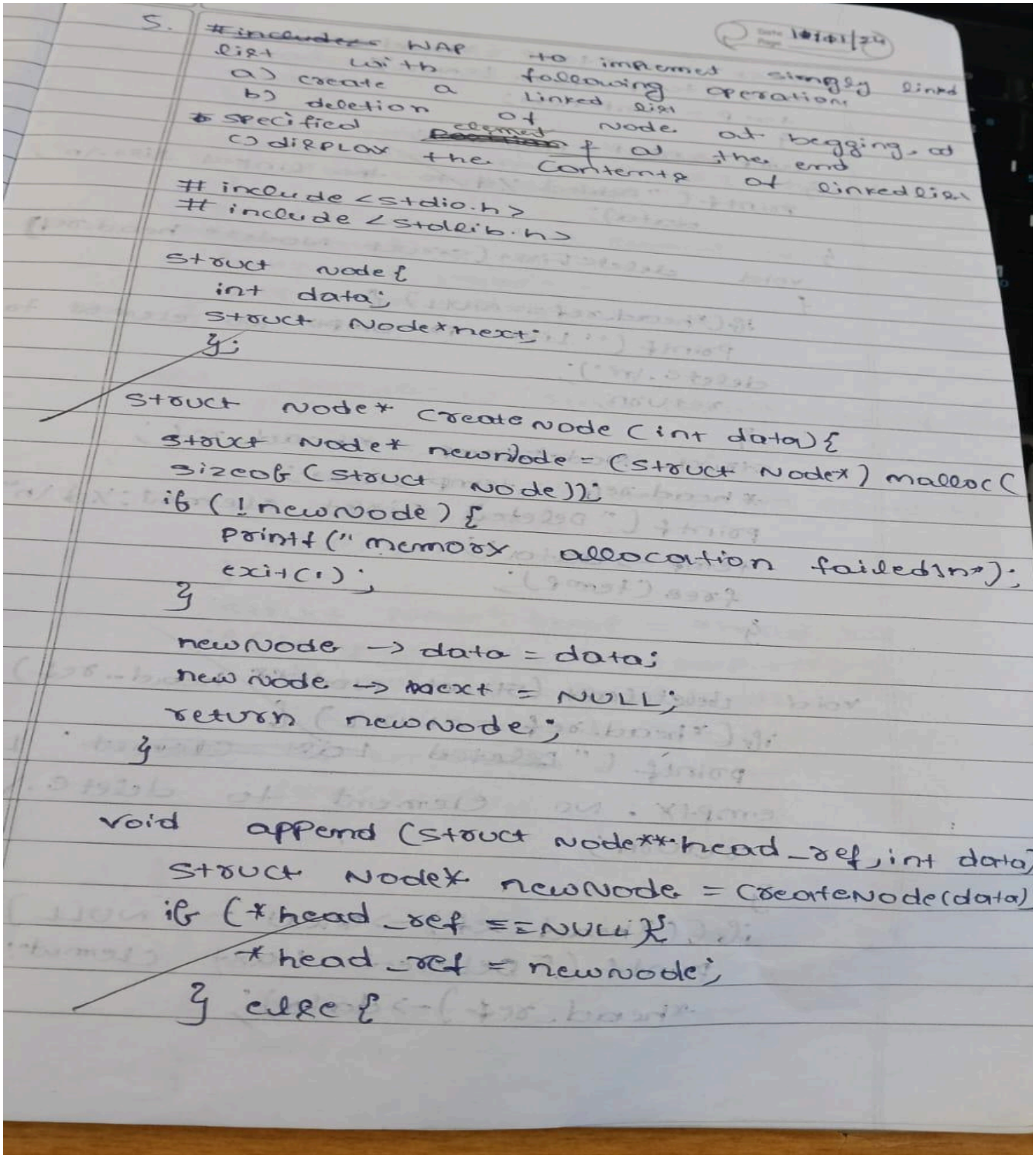
6.WAP to Implement Singly Linked List with following operations

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.

6b. Leetcode Problem -- 739 (Daily Temperature)



```
    struct Node* temp = *head_ref;
    while (temp->next != NULL){
      temp = temp->next;
    }
    temp -> next = newNode;
  }
  printf("Added %d to the linked list.\n",
        data);
}
void deleteFirst(struct Node** head_ref)
{
    if(*head_ref == NULL) {
      Printf("List is empty. No element to
    delete.\n");
      return;
    }
    struct Node* temp = *head_ref;
    *head_ref = temp->next;
    printf("Deleted first element: %d\n",
    temp->data);
    free(temp);
  }

void deleteLast(struct Node** head_ref) {
    if(*head_ref == NULL) {
      printf("Deleted Last element. List is
    empty. No element to delete.\n");
      return;
    }
    if((*head_ref)->next == NULL){
      printf("Deleted Last element: %d\n",
      *head_ref)->data);
```

```
free(*head_ref);
*head_ref = NULL;
return;
}
struct Node* temp = *head_ref;
while (temp -> next -> next != NULL) {
    temp = temp -> next;
}
Printf ("Deleted last element : %d\n", temp->
next -> data);
free(temp -> next);
temp -> next = NULL;
}

void deleteelement (struct Node** head_ref, int
    {
    if (*head_ref = NULL){
    Printf ("List is empty. No element to
        delete. \n");
    return;
    }
    struct Node* temp = *head_ref;
    if (temp != NULL && temp -> data == key)
        *head_ref = temp -> next;
        Printf ("Deleted specified element : %d
            key);
        free(temp);
        return;
}
Struct Node* prev = NULL;
while (temp != NULL && temp -> data != k
{   prev = temp;
    temp = temp -> next;
}
```

```
if(temp == NULL) {
printf("Element %d not found in the
list \n", key);
return;
}
prev->next = temp->next;
printf("Deleted specified element :%d\n", ke
free(temp);
}

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

nt main()
struct node*head = NULL;
int choice, value;
Printf("
while(1) {
Printf("(*) MENU:");
Printf("Enter 1 to add the element:");
printf("Enter 2 to delete at First:");
Printf("Enter 3 to delet, at Last:");
```

```
printf("Enter 4 to delete the element at
position:");
printf("Enter 5 to display the Linked list");
printf("Exit:");
printf("Enter your option:");
scanf("%d", &choice);
switch (choice)
{
case 1:
    printf("Enter value:");
    scanf("%d", &value);
    append(&head, value);

case 2:
    deleteFirst(&head);
case 3:
    deleteLast(&head);
case 4:
printf("Enter position:");
scanf("%d", &value);
deleteelement(&head, value);
case 5:
    display();
}
}
return 0;
}
```

Dynamic memory allocation

malloc
calloc

1. Insertion at beggining
2. Insertion at end
3. Deletion at beginning
4. Deletion at end
5. Deletion at Specified position
6. Display list

Enter your choice: 1
Enter data to insert at beggining: 11

Executed 11/11

Enter your choice: 2
Enter data to insert at end: 22

Enter your choice: 6
11 22

Enter your choice: 3
Data is deleted at beginning

Enter your choice: 4
Data is deleted at ending

Enter your choice: 6
List is empty

Enter your choice: 1
Enter data to insert at beggiu
2 2

Enter your choice: 5
Enter position to delete 0

Enter your choice 6
List is empty

Code:

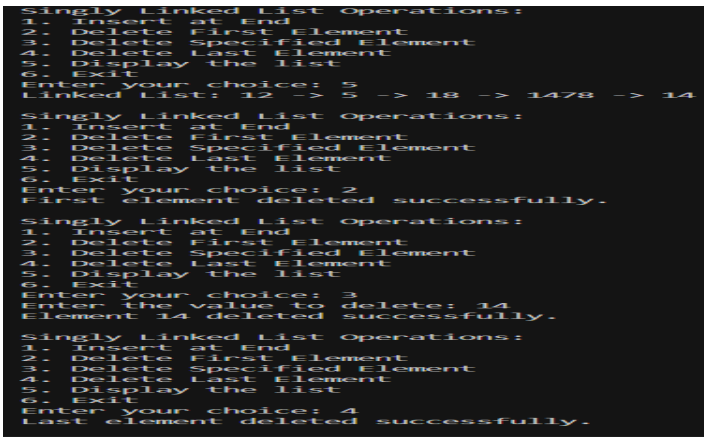
```
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a linked list node
struct Node {
    int data;
    struct Node* next;
};
// Function to create a linked list
void createList(struct Node** head) {
    *head = NULL;
}
// Function to delete the first element from the list
void deleteFirst(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("First element deleted successfully.\n");
}
// Function to delete a specified element from the list
void deleteElement(struct Node** head, int value) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
        return;
    }
    struct Node* temp = *head;
    struct Node* prev = NULL;
    // If the element to delete is at the head
    if (temp != NULL && temp->data == value) {
        *head = temp->next;
        free(temp);
        printf("Element %d deleted successfully.\n", value);
        return;
    }
    // Search for the element to delete
    while (temp != NULL && temp->data != value) {
        prev = temp;
```

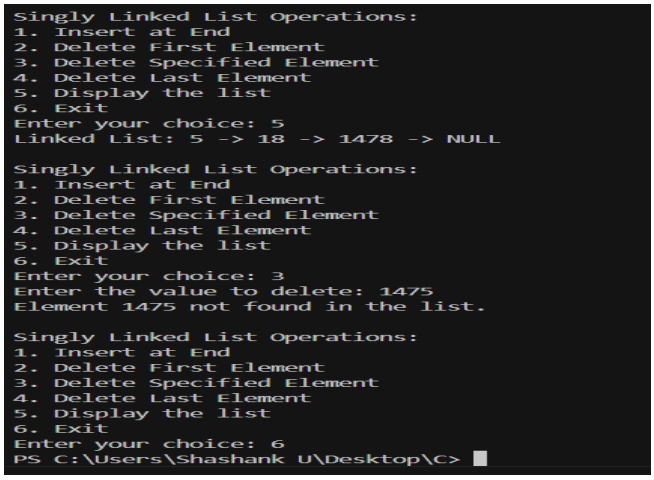
```
        temp = temp->next;
    }
    // If the element was not found
    if (temp == NULL) {
        printf("Element %d not found in the list.\n", value);
        return;
    }
    // Delete the node
    prev->next = temp->next;
    free(temp);
    printf("Element %d deleted successfully.\n", value);
}
// Function to delete the last element from the list
void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
        return;
    }
    // If there is only one node
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        printf("Last element deleted successfully.\n");
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    printf("Last element deleted successfully.\n");
}
// Function to display the contents of the linked list
void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The 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");
}
// Function to insert an element at the end of the list
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Node with value %d inserted at the end.\n", value);
}

// Main function
int main() {
    struct Node* head;
    createList(&head); // Create an empty list

    int choice, value;
    while (1) {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at End\n");
        printf("2. Delete First Element\n");
        printf("3. Delete Specified Element\n");
        printf("4. Delete Last Element\n");
        printf("5. Display the list\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert at the end: ");
```

```
            scanf("%d", &value);
            insertAtEnd(&head, value);
            break;

        case 2:
            deleteFirst(&head);
            break;

        case 3:
            printf("Enter the value to delete: ");
            scanf("%d", &value);
            deleteElement(&head, value);
            break;

        case 4:
            deleteLast(&head);
            break;

        case 5:
            displayList(head);
            break;

        case 6:
            exit(0);

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

Output:





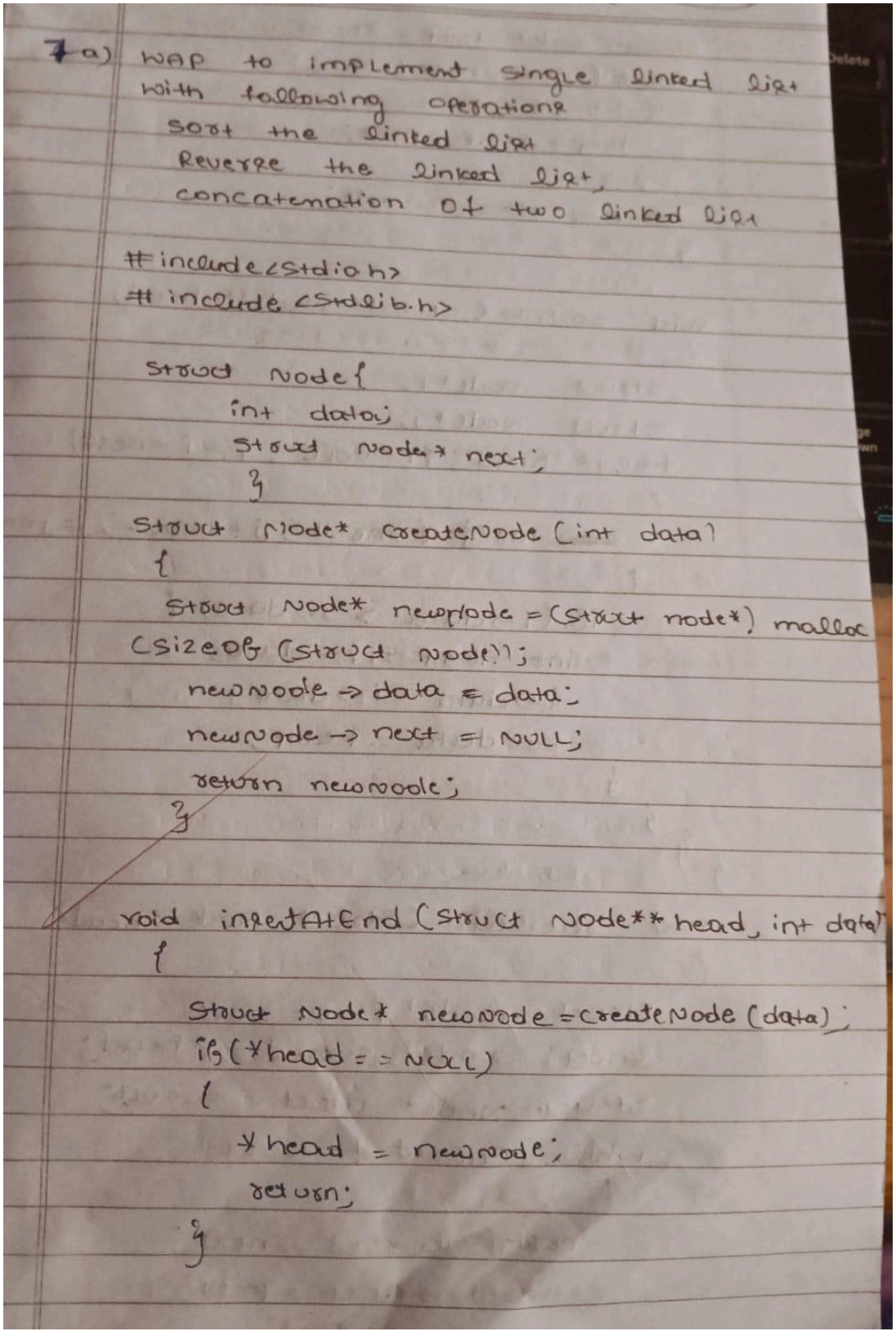
Leetcode problem(Daily Temperatures):

C Auto

```c
int* dailyTemperatures(int* temperatures, int temperaturesSize, int* returnSize) {
    *returnSize=temperaturesSize;
    int* answer=(int*)calloc(temperaturesSize,sizeof(int));
    int* stack = (int*)malloc(temperaturesSize * sizeof(int));
    int top = -1;

    for (int i = 0; i < temperaturesSize; i++) {
        while (top >= 0 && temperatures[i] > temperatures[stack[top]]) {
            int idx = stack[top--];
            answer[idx] = i - idx;
        }
        stack[++top] = i;
    }

    *returnSize = temperaturesSize;
    free(stack);
    return answer;
}
```

Problem 7:

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



```
7 a) WAP to implement single linked list
with following operations
  sort the linked list
  Reverse the linked list,
  concatenation of two linked list

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

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


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

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

void sortList (struct node** head)
{
  struct node* i;
  struct node* j;
  for(i=*head; i!=NULL; i=i->next)
  {
    for(j=i->next; j!=NULL; j=j->next
    {
      if (i->data > j->data) {
        int temp = i->data;
        i->data = i->data;
        i->data = temp;
      }
    }
  }
}

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

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

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

int main()
{
printf("\n 1. Insert into list1. \n 2. Insert
into List2 \n Sort list1 \n4. Reverse list1 \n
```

```
5. concatenate LinkedList. DisplayList Ins. Exit
In Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
 case 1:
  printf("Enter data to insert into list 1:");
  scanf("%d", &data);
  insertAtEnd(&list1, data);
  break;
 case 2:
  printf("Enter data to insert into List 2:");
  scanf("%d", &data);
  insertAtEnd(&list2, data);
  break;
 case 3:
  sortList(&list1);
  printf("List 1 sorted.\n");
  break;
 case 4:
  reverseList(&list1);
  printf("List 1 reversed.\n");
  break;
 case 5:
  concatenateLists(&list1, &list2);
  printf("Lists concatenated.\n");
  break;
 case 6:
  printf("List 1:");
  displayList(list1);
  break;
```

```
case 7:
    printf ("List 2: ");
    displayList(list2);
    break;
case 8:
    exit(0)
default:
    printf ("Invalid choice. Try again.\n");
}
}
return 0;
}
```

O/P

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse list 1
5. Concatenate List
6. Display List 1
7. Display List 2.
8. Exit

Enter your choice : 1
Enter data to insert into List 1 : 32
Enter your choice : 1
Enter data to insert into List 1 : 23
Enter your choice : 2
Enter data to insert into List 2 : 11
Enter your choice : 2
Enter data to insert into List 2 : 34
Enter your choice :
List 1 sorted

~~4. Reverse List~~

Enter your choice : 6

List 1 : 23 322

Enter your choice : 4

List 1 reversed.

List 1 : 322 23

Enter your choice : 5

List 1 concatenated.

Enter your choice : 6

List 1 : 322 23 11 34.

5 b. WAP to implement single link list to simul

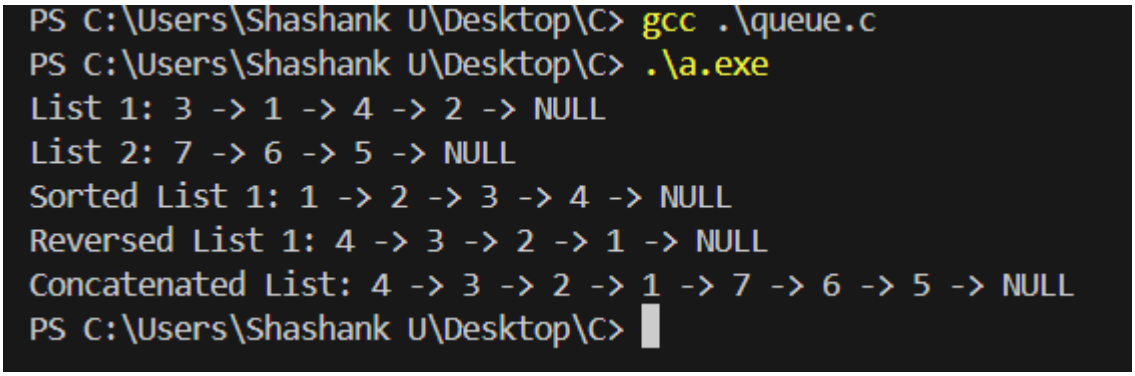
Code:

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

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

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

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

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

void sortList(struct Node** head) {
    if (*head == NULL || (*head)->next == NULL)
        return;
    struct Node* i, *j;
    int temp;
```

```
    for (i = *head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

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

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

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    insertAtEnd(&list1, 3);
    insertAtEnd(&list1, 1);
    insertAtEnd(&list1, 4);
```

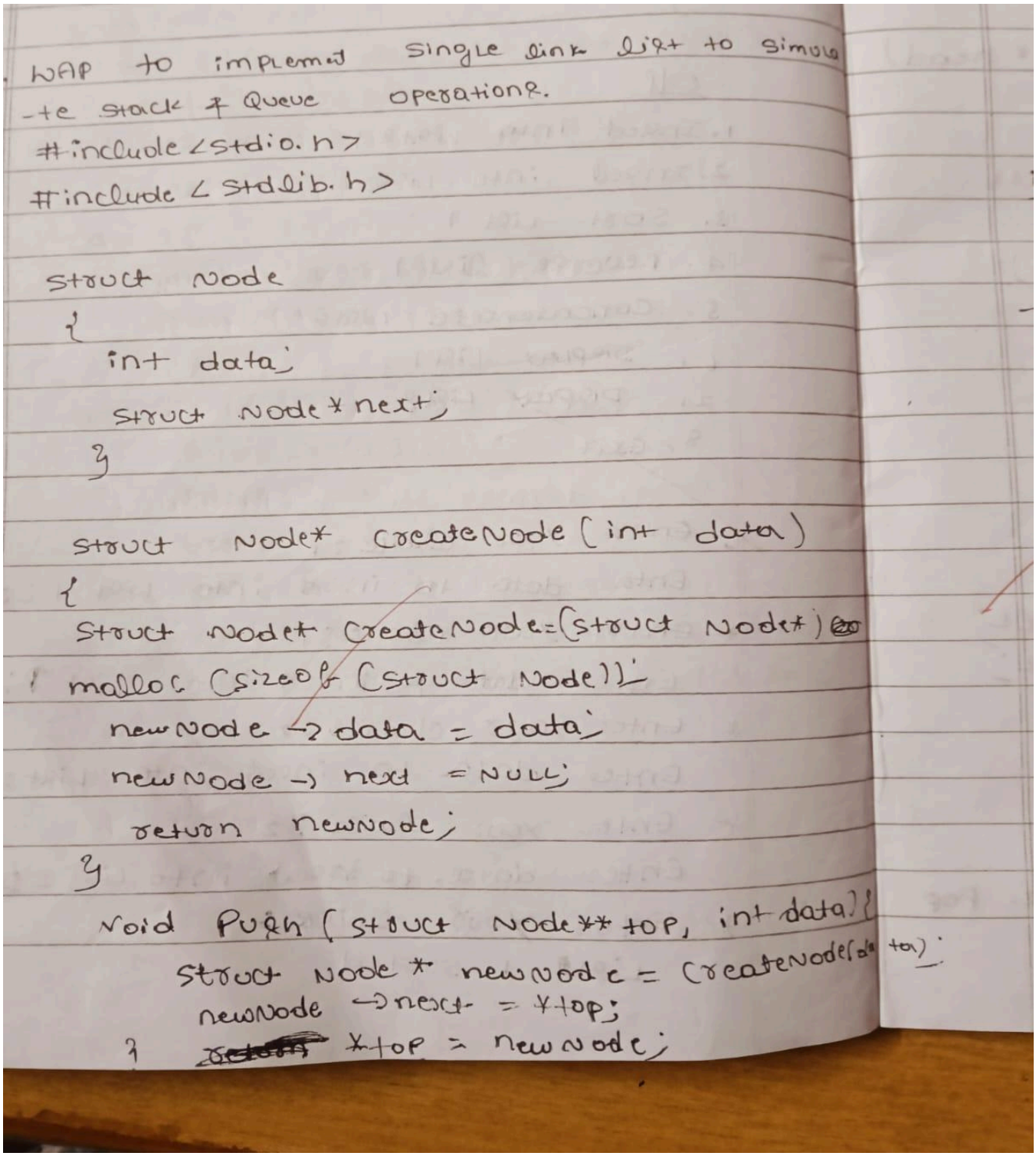
```
    insertAtEnd(&list1, 2);

    insertAtEnd(&list2, 7);
    insertAtEnd(&list2, 6);
    insertAtEnd(&list2, 5);

    printf("List 1: ");
    printList(list1);
    printf("List 2: ");
    printList(list2);

    sortList(&list1);
    printf("Sorted List 1: ");
    printList(list1);

    reverseList(&list1);
    printf("Reversed List 1: ");
    printList(list1);

    concatenateLists(&list1, &list2);
    printf("Concatenated List: ");
    printList(list1);

    return 0;
}
```

Output:



Problem 8:

WAP to Implement Singly Linked List to simulate Stack Operations.



```
WAP to implement single link list to simula
-te stack & Queue operations.
#include <stdio.h>
#include <stdlib.h>

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

struct Node* createNode (int data)
{
struct Node* createNode=(struct Node*)
malloc (sizeof (struct Node));
   newNode -> data = data;
   newNode -> next = NULL;
   return newNode;
};
void Push (struct Node** top, int data){
      struct Node * newNode = (createNode(da
      newNode ->next = *top;
}      *top = newNode;
```

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

void displayList(struct Node* head)
{
  struct Node *temp = head;
  while (temp != NULL)
  {
    printf("%d", temp->data);
    temp = temp->next;
  }
  printf("\n");
}
int main()
{
  struct node* Stack = NULL;
  struct Node* QueueFront = NULL;
  struct Node* QueueRear = NULL;
  int choice, data;

while(1)
{
  printf("\n1. Push to Stack \n2. Pop
```

from stack in 3. Display stack no. enQue
to Queue in 5. DeQueue from Queue in 7. Exit
Enter your choice:");
scanf("%d", &choice);

16/12

switch (choice) {
case 1:
    print ("Enter data to PUSH to stack");
    scanf ("%d", &data1);
    PUSH (&stack, data);
    break;
case 2:
    printf ("Popped from stack: %d \n", POP
    &stack));
    break;
case 3:
    printf ("stack 3:");
    displaylist (stack);
case 4:
    Printf ("Enter data to enQueue to
    Queue:");
    scanf ("%d", &data);
    enQueue (& QueueFront, & QueueRear, data
case 5:
    Printf ("DeQued from Queue: %d\n", de
    -Queue (&QueueFront, &QueueRear);
    break;
case 6:
    Printf ("Queue:");
    displayList (QueueFront);
    break;
case 7:
    exit(0);

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

Output

```
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: 11
Enter your choice: 1
Enter data to push to stack: 22
Enter your choice: 3
Stack: 22 11
Enter your choice: 2
Popped from stack: 22
Enter your choice: 3
Stack: 11
Enter your choice: 4
Enter data to enqueue to Queue: 56
Enter your choice: 4
Enter data to enqueue to Queue: 45
Enter your choice: 5
Dequeued from Queue: 56
Enter your choice: 6
Queue: 45
```

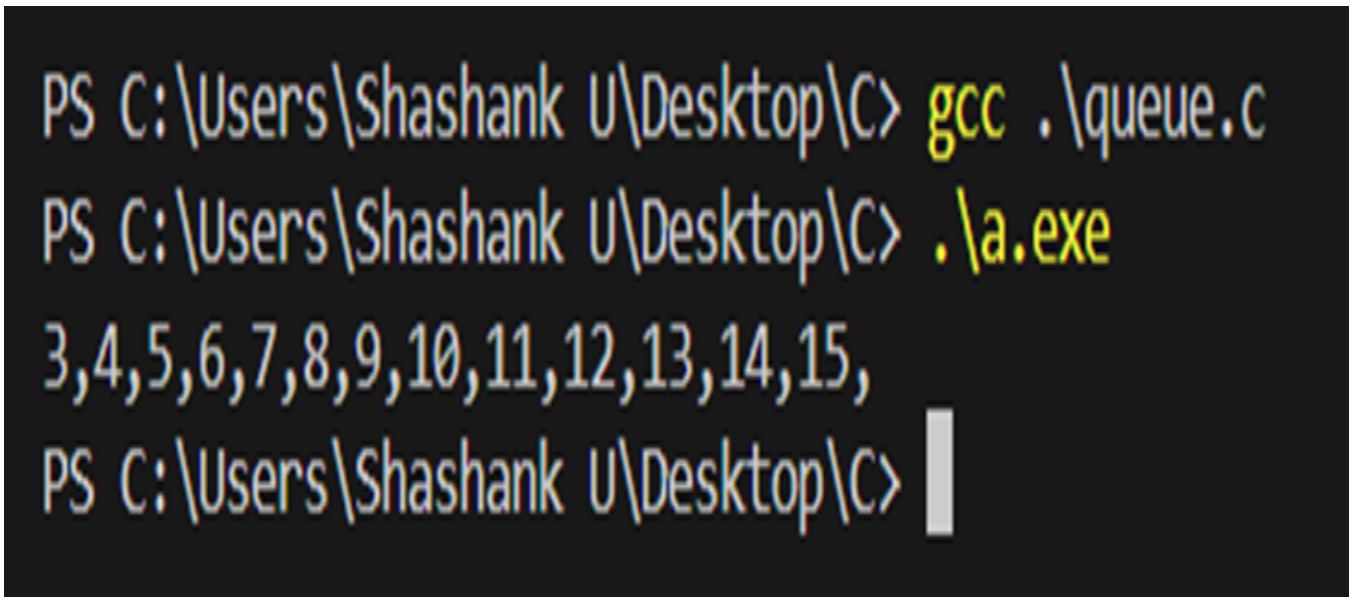
Code:

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

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

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

struct Node* top=NULL;

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

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

void peek(){
        if(top==NULL){
        printf("Empty stack!\n");
        return;
        }
        printf("Peek element:%d\n",top->data);
}

void display(){
        if(top==NULL){
        printf("Empty stack\n");
        return;
        }
        for(struct Node*temp=top;temp!=NULL;temp=temp->next){
```

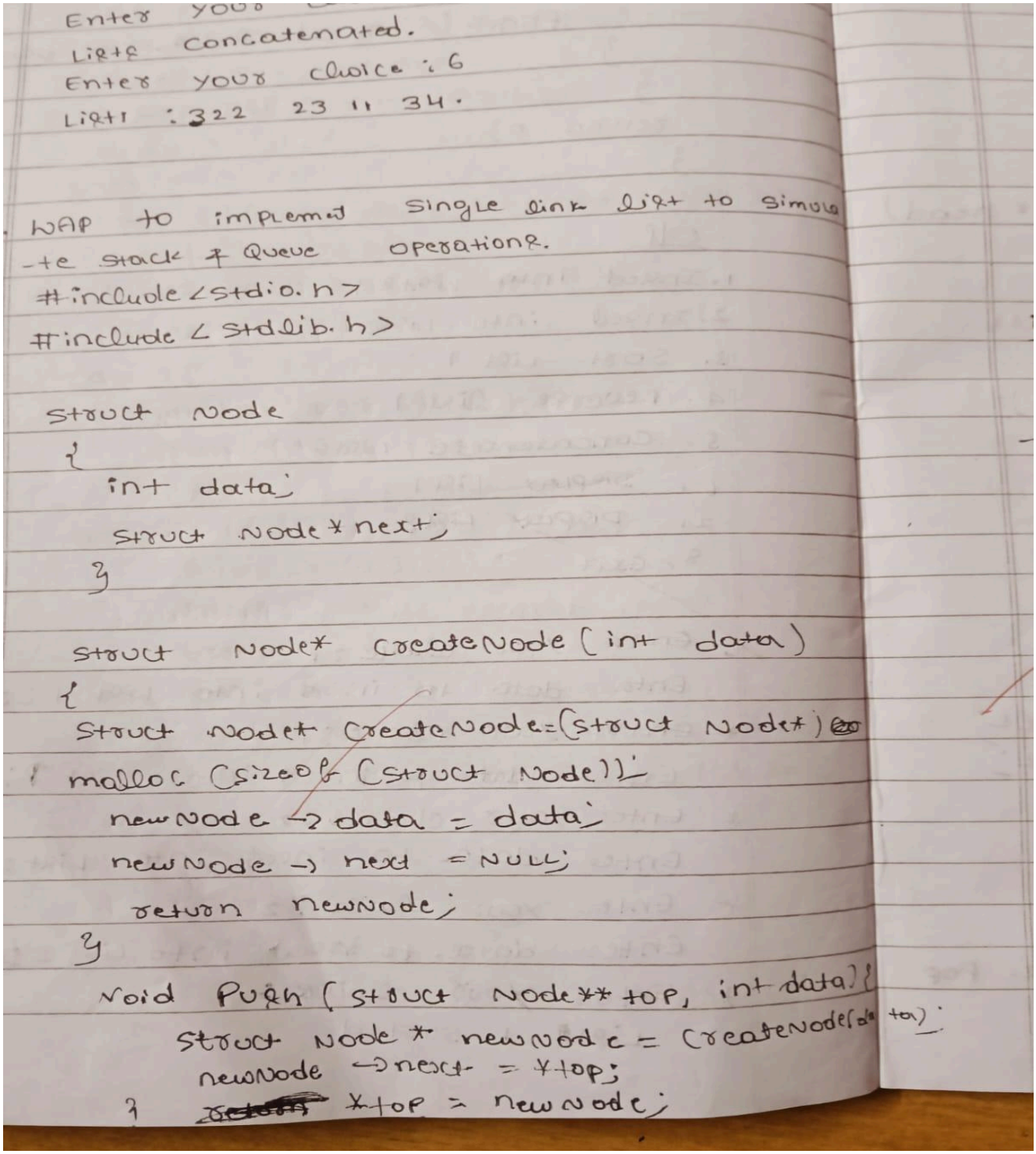
```
        printf("%d,",temp->data);
        }
        printf("\n");
}

int main(){
        for(int i=15;i>0;i--){
        push(i);
        }
        pop();
        pop();
        display();
        return 0;
}
```

Output:



Problem 9:

WAP to Implement Single Link List to simulate Queue Operations.



```
Enter your
List concatenated.
Enter your choice : 6
List : 322 23 11 34.

WAP to implement single link list to simula
-te stack & Queue operations.
#include <stdio.h>
#include <stdlib.h>

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

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

```
if POP( struct Node** top, int data) {
  struct Node* newNode = CreateNode
```

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

```
void enQueue( struct Node** front, struct Node
-e** rear, int data)
  {
    struct Node* newNode = CreateNode(data);
    if (*rear == NULL){
    *front = *rear = newNode;
      return;
    }
    (*rear)->nest = newNode;
    *rear = newNode;
  }
```

```
int dequeue (struct Node** front, struct Node**
rear){
  if (*front == NULL)
    {
      Printf ("Queue underflow \n");
```

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

void displaylist(struct Node* head)
{
  struct Node *temp = head;
  while (temp != NULL)
  {
    printf("%d", temp->data);
    temp = temp->next;
  }
  printf("\n");
}
int main()
{
  struct node* Stack = NULL;
  struct Node* Queuefront = NULL;
  struct Node* QueueRear = NULL;
  int choice, data;

  while(1)
  {
  printf("\n1. Push to Stack \n 2. Pop
```

```
from stack in 3. Display stack no. enQue
to Queue in 5. DeQueue from Queue in 7. Exit
Enter you o choice:");
Scanf("%d", &choice);

Switch (choice) {
case 1:
    Print ("Enter data to PUSH to stack")
    Scanf ("%d", &data1);
    PUSH (&stack, data1);
    break;
case 2:
    Printf ("Popped from stack: %d \n", POP
    &stack));
    break;
case 3:
    Printf ("Stack 3:");
    displayList (stack);
case 4:
    Printf ("Enter data to enQueue to
    Queue:");
    scanf ("%d", &data);
    enQueue (& QueueFront, & QueueRear, da
case 5:
    Printf ("DeQued from Queue: %d\n",
    -Queue (&QueueFront, &QueueRear));
    break;
case 6:
    Printf ("Queue:");
    displayList (QueueFront);
    break;
case 7:
    exit(0);
```

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

Output

```
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: 11
Enter your choice: 1
Enter data to push to stack: 22
Enter your choice: 3
Stack: 22 11
Enter your choice: 2
Popped from stack: 22
Enter your choice: 3
Stack: 11
Enter your choice: 4
Enter data to enqueue to Queue: 56
Enter your choice: 4
Enter data to enqueue to Queue: 45
Enter your choice: 5
Dequeued from Queue: 56
Enter your choice: 6
Queue: 45
```

Code:

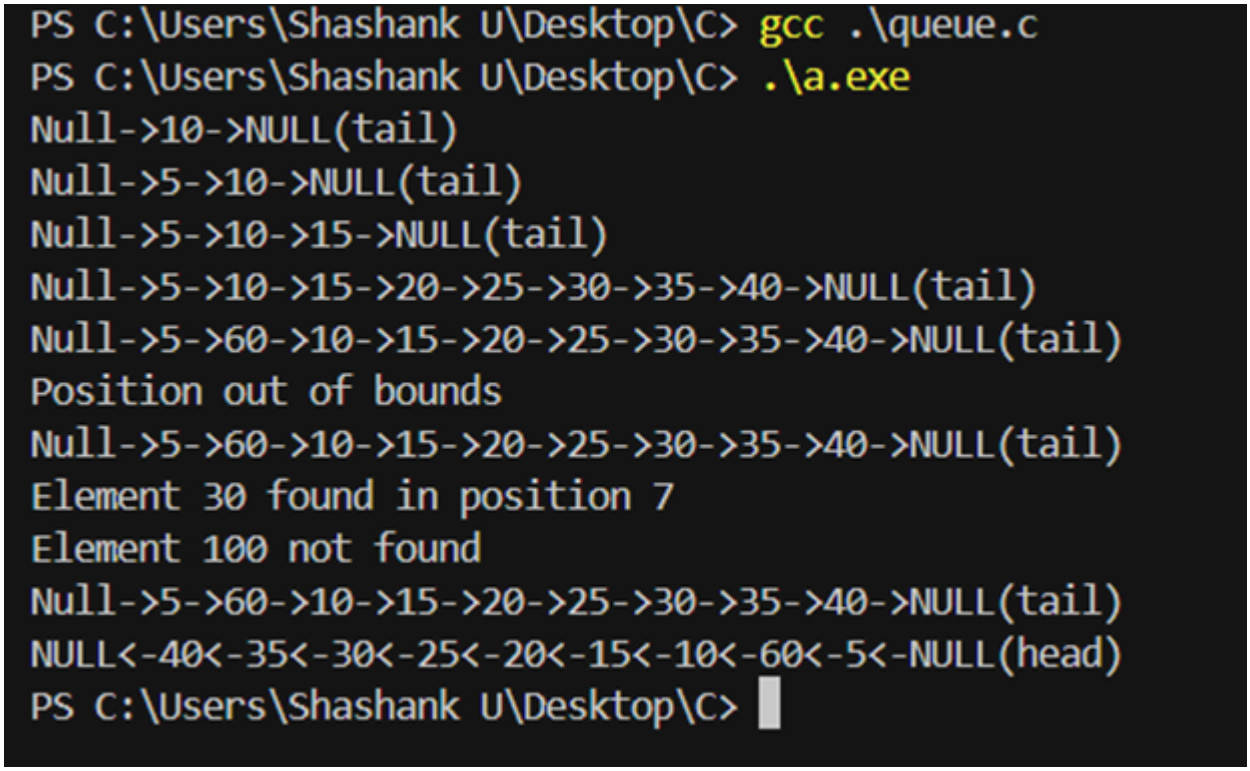
```
#include<stdio.h>
#include<stdlib.h>

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

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

struct Node *front=NULL,*rear=NULL;

void enqueue(int value){
    struct Node*newNode=createNode(value);
    if(front==NULL && rear==NULL){
        front=newNode;
        rear=newNode;
        return;
```

```
    }
    rear->next=newNode;
    rear=newNode;
}
void deque(){
    if(front==NULL && rear==NULL){
       printf("Queue is empty");
       return;
    }
    struct Node*temp=front;
    front=front->next;
    free(temp);
}
void front_(){
    if(front==NULL && rear==NULL){
       printf("Queue is empty");
       return;
    }
    printf("front element:%d\n",front->data);
}

void rear_(){
   if(front==NULL && rear==NULL){
       printf("Queue is empty");
```

```
        return;
    }
    printf("rear element:%d\n",rear->data);
}

void display(){
    if(front==NULL && rear==NULL){
        printf("Queue is empty");
        return;
    }
    for(struct Node*temp=front;temp!=NULL;temp=temp->next){
        printf("%d,",temp->data);
    }
    printf("\n");
}

int main(){
    for(int i=1;i<16;i++){
        enqueue(i);
    }
    deque();
    deque();
    front_();
    rear_();
```

```
    display();
    return 0;
}
```

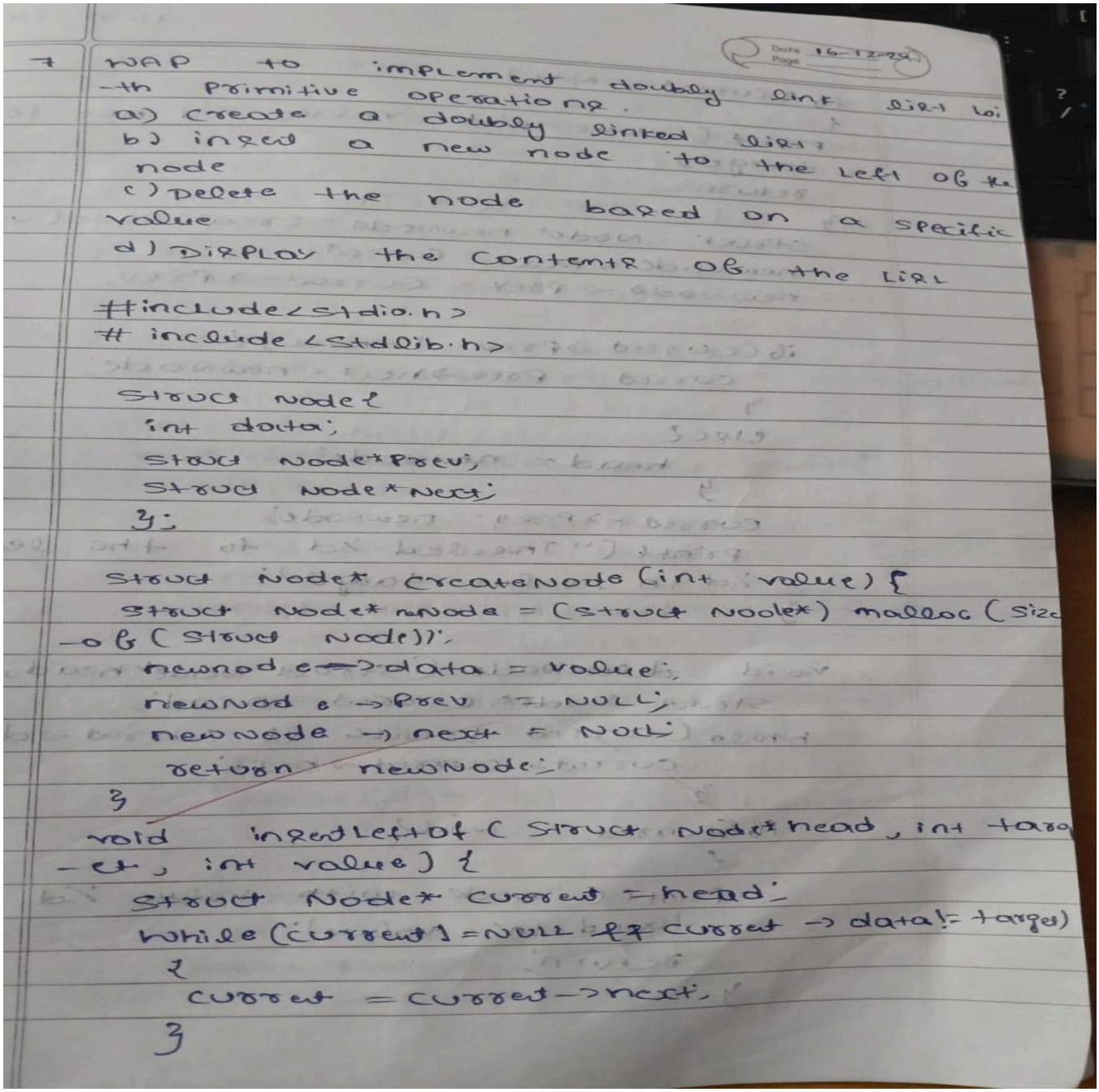
Output:



Problem 10;

WAP to Implement doubly link list with primitive operations

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



```
7 WAP to implement doubly link list
-th primitive operations.
a) create a doubly linked list
b) insert a new node to the left of t
node
c) Delete the node based on a specific
value
d) Display the contents of the list

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

struct node{
 int data;
 struct Node* Prev;
 struct Node * Next;
};

struct Node* CreateNode (int value) {
 struct Node* newNode = (struct Node*) malloc (size
-of (struct Node));
 newnode -> data = value;
 newNode -> Prev = NULL;
 newNode -> next = NULL;
  return newNode;
}
void insertLeftOf ( struct Node* head, int targ
-et, int value ) {
 struct Node* current = head;
 while (current != NULL && current -> data != target)
 {
  current = current -> next;
 }
```

16-12-24

```
if (current == NULL)
{
printf ("Node with value %d not found
target);
return;
}
struct Node* newNode = createNode(value);
newNode->next = current;
newNode->prev = current->prev;

if(current->prev != NULL) {
  current->prev->next = newNode;
}
else{
    head = newNode;
}
current->prev = newNode;
printf("Inserted %d to the left of
%d \n", value, target);
}

void deleteNode(struct Node* head, int val
  struct Node* current = head;
while (current != NULL && current->data != val
    current = current->next;
}
if(current == NULL)
{
```

```
if(current->prev != NULL)
 {
 current->prev->next = current->next;
 }
 else{
 head = current->next
  }
if(current->next != NULL)
 { current->next->prev = current->prev;
 }
 free(current);
 Printf("Deleted node with value %d\n", va
  -e);
}
void display (struct Node* head) {
    if(head==NULL)
      { printf("List is empty:\n");
        return;
       }
    struct Node* current = head;
    Printf("Doubly linked list:");
    while(current!=NULL){
     printf("%d", current->data);
      current = current->next;
     }
    Printf("\n");
}

int main()
```

```
while(choice) {
  case 1:
      Printf("Enter the target value to
insert left of:");
      scanf("%d", &target);
      Printf("Enter the value to insert:");
      scanf("%d", &value);
      insertLeftOf(head, target, value);
      break;
  case 2:
      Printf("Enter the value of the n
-de to delete:");
      scanf("%d", &value);
      deleteNode(head, value);
      break;
case 3:
      displar(head);
      break;
case 4:
      Printf("Exiting program...\n");
      exit(0);
default:
      Printf("Invalid choice. Try again.
  }
}
return 0;
}
```

Date 16-12-24
Page

O/P

Doubly linked list operations
1. Insert Left of Node
2. Delete Node by value
3. Display list
4. Exit

Enter your choice: 1
Enter the target value to insert Left o
10
Node with value 10 not found.

Enter your choice: 1
Enter the value of the node to dele
Node with value 5 not found

Enter your choice: 3
List is empty

Enter your choice: 4
Exit.

Code:

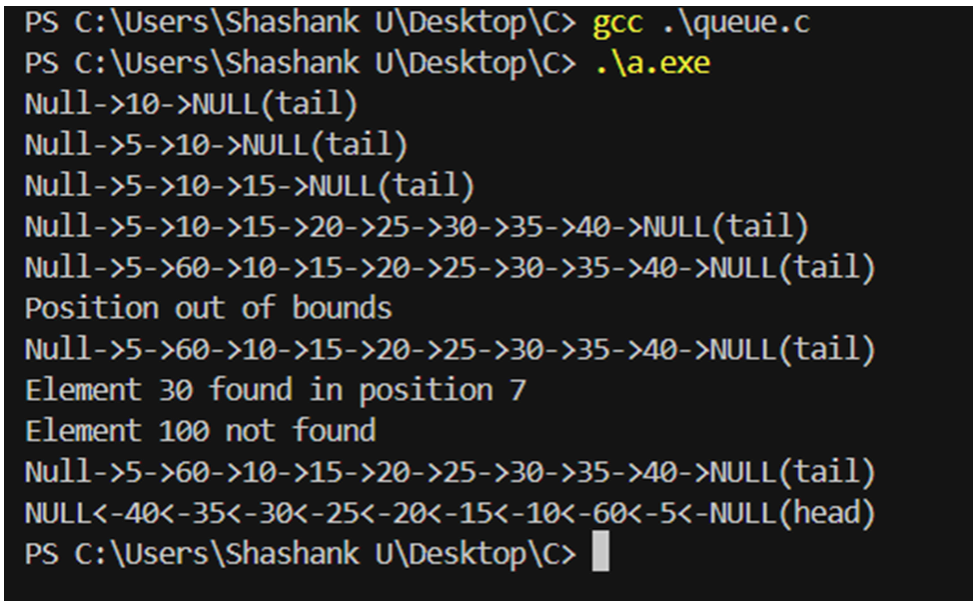
```
#include<stdio.h>
#include<stdlib.h>

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

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

void insert_at_begin(struct Node**head,struct Node**tail,int value){
	struct Node*node=createNode(value);
	if(*head==NULL){
	*head=*tail=node;
	return;
	}
```

```
    node->next=*head;
    (*head)->prev=node;
    *head=node;
}

void insert_at_end(struct Node**head,struct Node**tail,int value){
    struct Node*node=createNode(value);
    if(*head==NULL){
    *head=*tail=node;
    return;
    }
    (*tail)->next=node;
    node->prev=*tail;
    *tail=node;
}

void insert_at_pos(struct Node**head,struct Node**tail,int pos,int value){
    struct Node*node=createNode(value);
    if(*head==NULL || pos==1){
    insert_at_begin(head,tail,value);
    return;
    }
    if(pos<=0){
    printf("Invalid position\n");
```

```
        return;
        }
        struct Node* temp=*head;
        for(int i=1;i<pos-1;i++){
        if(temp==NULL){
        printf("Position out of bounds\n");
        free(node);
        return;
        }


int main(){
        struct Node *head=NULL,*tail=NULL;
        insert_at_begin(&head,&tail,10);
        display(&head);
        insert_at_begin(&head,&tail,5);
        display(&head);
        insert_at_end(&head,&tail,15);
        display(&head);
        insert_at_end(&head,&tail,20);
        insert_at_end(&head,&tail,25);
        insert_at_end(&head,&tail,30);
        insert_at_end(&head,&tail,35);
        insert_at_end(&head,&tail,40);
```

```
    display(&head);
    insert_at_pos(&head,&tail,2,60);
    display(&head);
    insert_at_pos(&head,&tail,80,6);
    display(&head);
    search(&head,30);
    search(&head,100);
    display(&head);
    reverse_print(&tail);
    return 0;
}
```

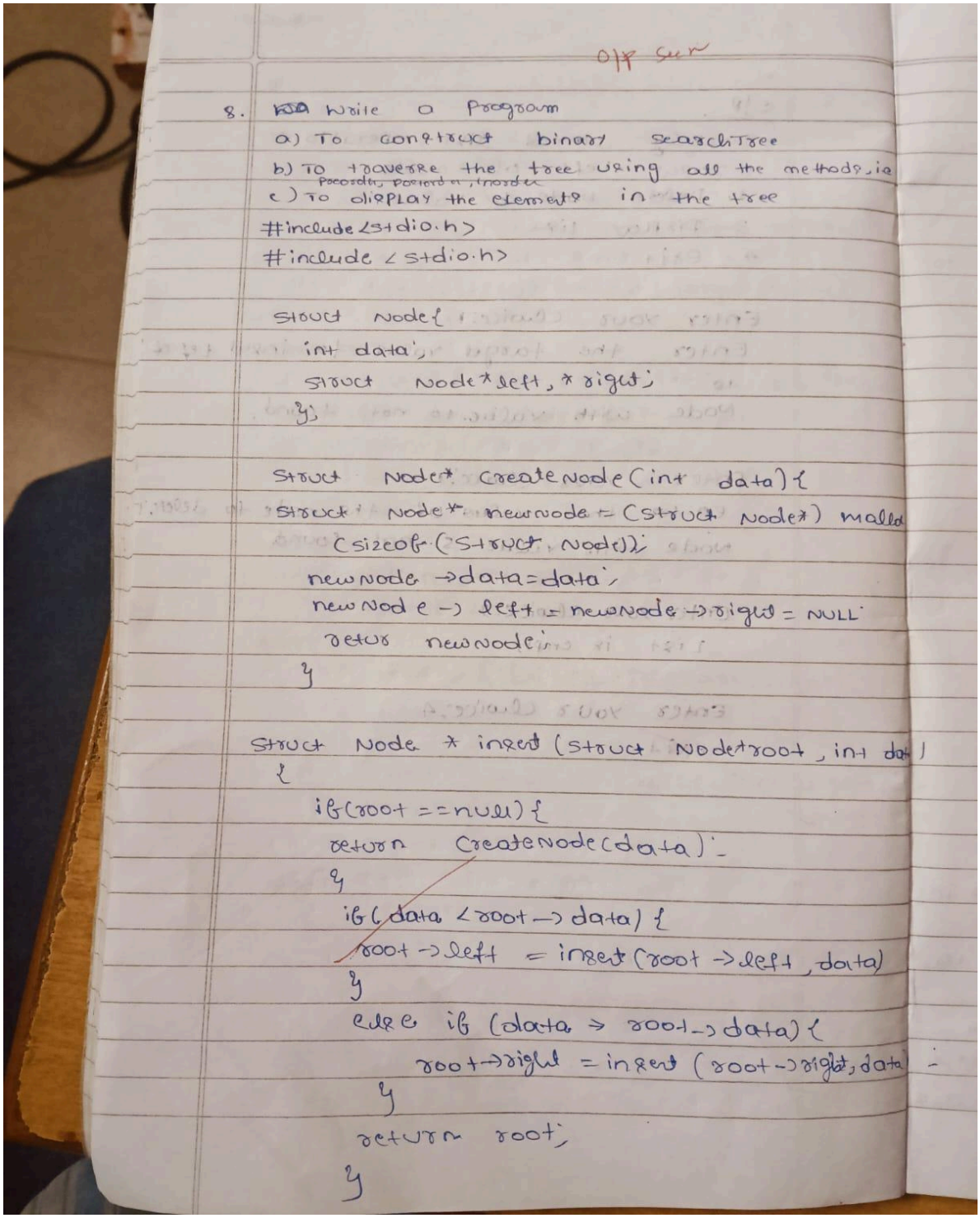
Output:



Problem 11 :Write a program a) ToconstructabinarySearchtree.

b) To traverse the tree using all the methods i.e., inorder, preorder and post order

c) To display the elements in the tree.



```
8. Write a program
a) To construct binary searchTree
b) To traverse the tree using all the methods, ie
   preorder, postorder, inorder
c) To display the elements in the tree
#include<stdio.h>
#include<stdio.h>

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

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

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

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

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

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

void displayTree (struct Node* root) {
    printf("Inorder Traversal:");
    inorder(root);
    Printf("\n");

    Printf("Preorder Traversal:");
    Preorder(root);
    printf("\n");

    Printf("Postorder Traversal:");
    postorder(root);
```

```
}

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

  Printf("Enter the number of elements to
  insert in the treee:");
  scanf("%d", &n);

  Printf("Enter the elements: \n");
    for(int i=0; i<n; i++){
      scanf("%d", &value);
      root = insert(root, value);
    }

  printf("Tree Traversals: \n");
  displayTree(root);

  return 0;

}
```

O/P

Binary Search Tree Operations

1. ~~Insert~~ Enter the number of elements to ins
2. 1 -ert in the treee:

Enter the elements: 12, 13, 14, 8, 9, 10,

Inorder Traversal: 8, 9, 10, 12, 12, 13 14

Pre-order Traversal: 12, 8, 9, 10, 12, 13, 14

Post-order Traversal: 10, 9, 8, 12, 14, 13, 12

Code

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

// Structure to represent a node in the BST
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a value into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

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

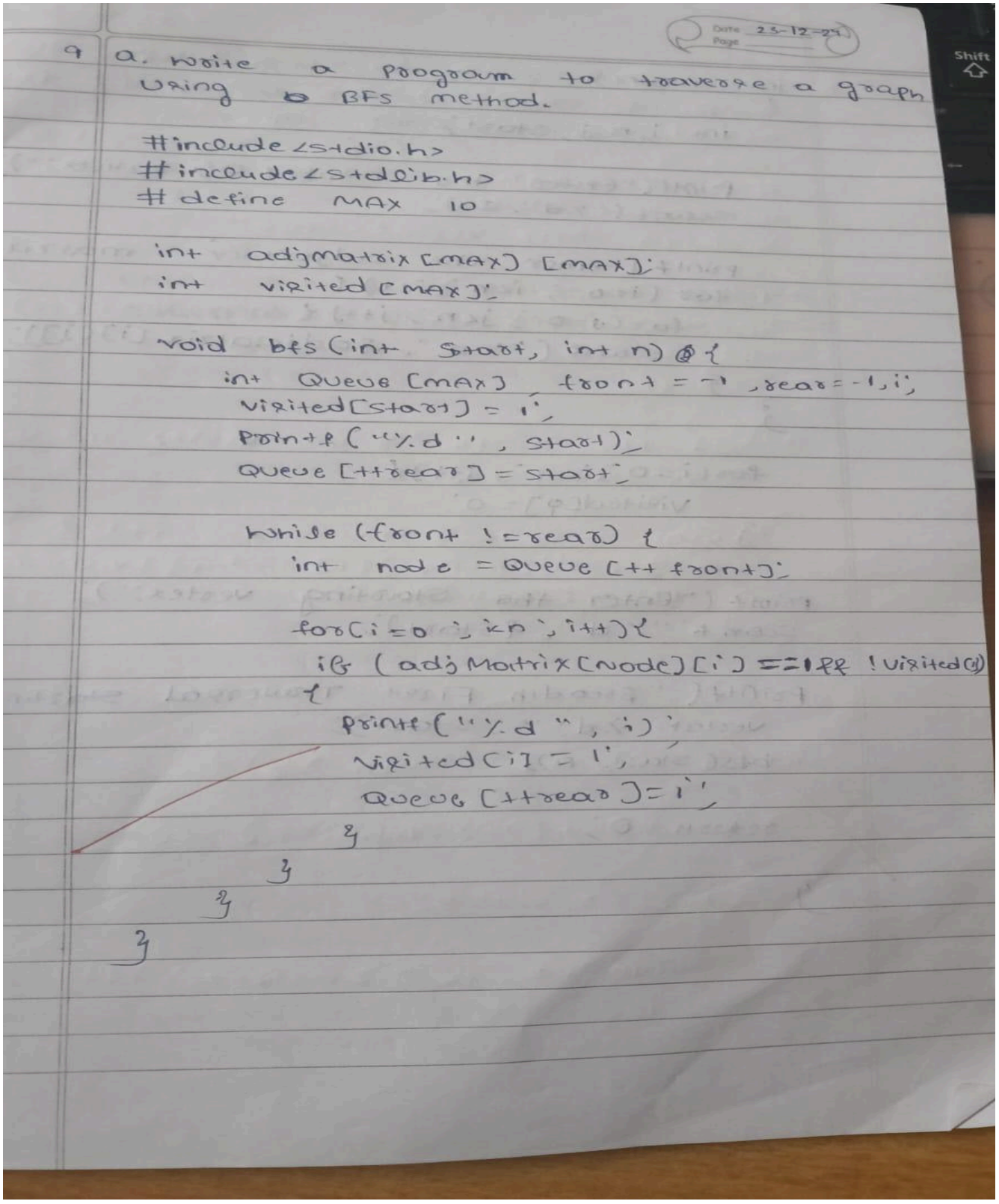
```
// Function for preorder traversal
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

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

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

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

    printf("Enter the elements: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");

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

    printf("Postorder traversal: ");
    postorder(root);
    printf("\n");
```

```
    return 0;
}
```

Output

```
PS C:\Users\User\Desktop> gcc 8th.c
PS C:\Users\User\Desktop> .\a.exe
Enter the number of elements in the tree: 5
Enter the elements:
3
334
55
12
23
Inorder traversal: 3 12 23 55 334
Preorder traversal: 3 334 55 12 23
Postorder traversal: 23 12 55 334 3
```

poblem 11
Write a program to traverse a graph using BFS method



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

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int adjMatrix[MAX][MAX];
int visited[MAX];

void bfs(int start, int n){
    int Queue[MAX], front=-1, rear=-1, i;
    visited[start]=1;
    printf("%d", start);
    Queue[++rear]=start;

    while(front!=rear){
        int node=Queue[++front];

        for(i=0;i<n;i++){
            if(adjMatrix[node][i]==1 && !visited[i])
            {
                printf("%d", i);
                visited[i]=1;
                Queue[++rear]=i;
            }
        }
    }
}
```

```
int main()
{
    int i,n,j, start;

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

    Printf("Enter the adjacency matrix:");
    for (i=0 ; i<n ; i++) {
        for(j=0 ; j<n ; j++) {
            scanf("%d", &adjmatrix[i][j]);
        }
    }

    for(i=0;i<n ; i++){
        visited[i]=0;
    }

Printf("Enter the starting vertex:");
 scanf "%d", &start);

Printf("Breadth First Traversal starting
  vertex %d:", start);
  bfs(start, n);

  return 0;

}
```

Date 31.2.24
Page

O/P

Enter the number of vertices: 3
Enter the adjacency matrix:

0 1
1 2
2 3
3 4
4 5

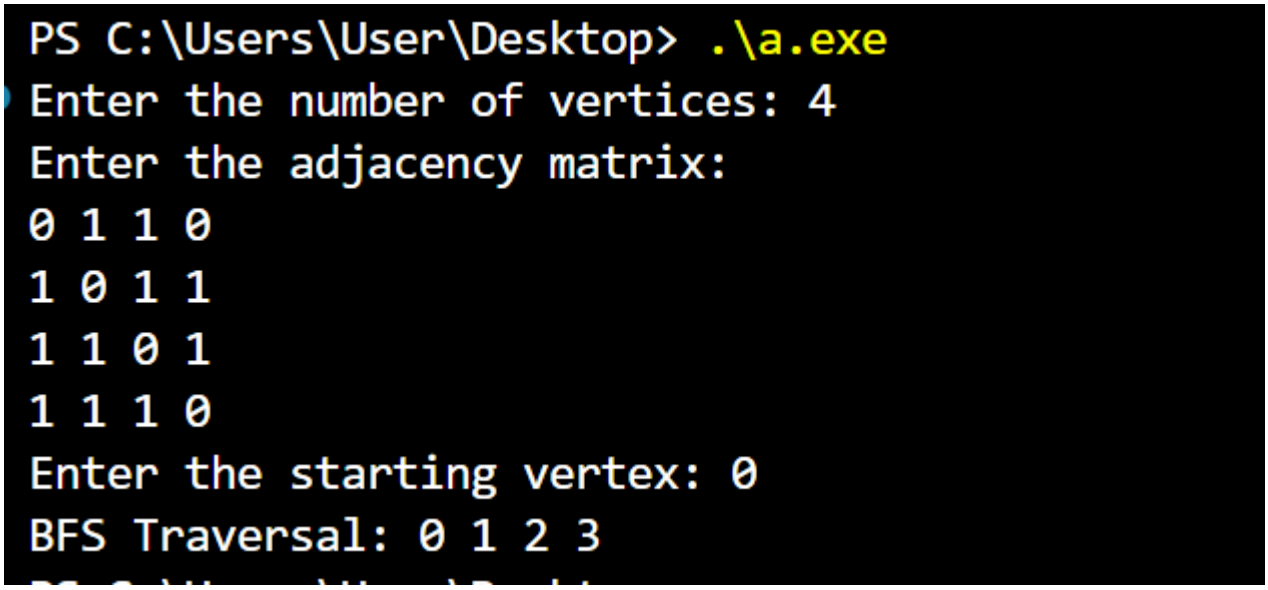
Enter the starting vertex: Breadth First Traversal starting from vertex 5 : 5

Code

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

#define MAX 100

int adj[MAX][MAX], visited[MAX], n;

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

int isConnected() {
    for (int i = 0; i < n; i++) visited[i] = 0;

    dfs(0);

    for (int i = 0; i < n; i++) {
        if (!visited[i]) return 0;
    }
    return 1;
}

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

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }

    if (isConnected()) {
        printf("The graph is connected.\n");
```

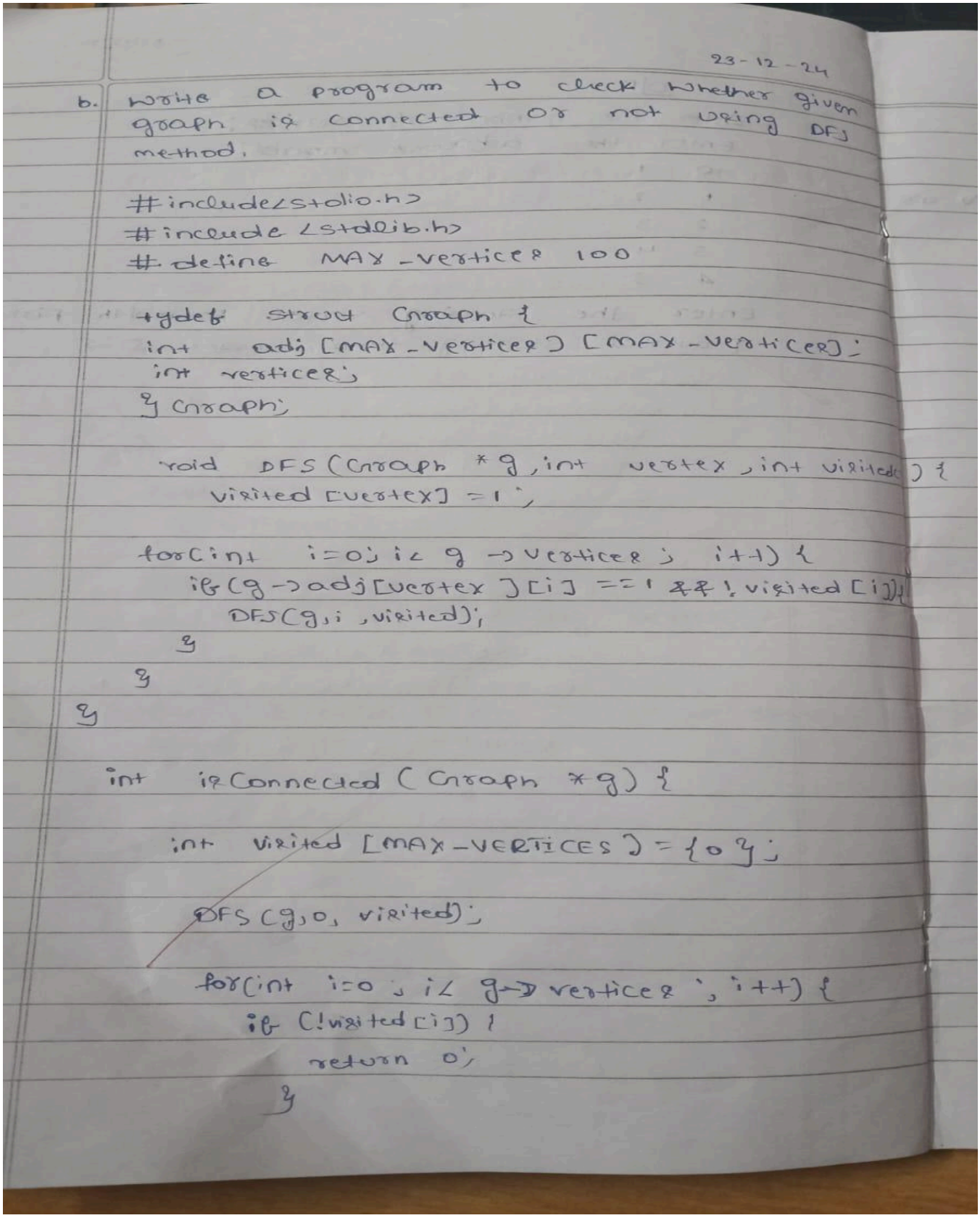
```
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

Output



Program 12

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



23-12-24

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

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_VERTICES 100

typedef struct Graph {
    int adj[MAX_VERTICES][MAX_VERTICES];
    int vertices;
} Graph;

void DFS(Graph *g, int vertex, int visited[]) {
    visited[vertex] = 1;

    for(int i=0; i< g->vertices; i++) {
        if(g->adj[vertex][i] == 1 && !visited[i]) {
            DFS(g, i, visited);
        }
    }
}

int isConnected(Graph *g) {

    int visited[MAX_VERTICES] = {0};

    DFS(g, 0, visited);

    for(int i=0; i< g->vertices; i++) {
        if(!visited[i]) {
            return 0;
        }
```

```
}
return 1;
}
int main() {
    Graph g;
    int edges, u, v;

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

    for(int i=0; i< g.vertices; i++)
    {
        for(int j=0; j< g.vertices; j++) {
            g.adj[i][j]=0;
        }
    }
    printf("Enter the number of edges:");
    scanf("%d", &edges);

    printf("Enter the edges (u,v) where u &
    v are vertex indices:\n");
    for(int i=0; i<edges; i++){
        scanf("%d%d", &u, &v);
        g.adj[u][v] = 1;
        g.adj[v][u] = 1;
    }
    if(isConnected(&g)) {
        printf("The graph is connected.\n");
    }
    else{
        printf("The graph is not connected.");
    }
    return 0;
}
```

23-12-24

output

Enter the number of vertices: 5

Enter the number of edges: 4

Enter the edges (U, V) where U and V are vertex indices:

0 1

1 2

2 3

3 4

The graph is connected

24/12

Code

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

#define MAX 100

int adj[MAX][MAX], visited[MAX], n;

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

int isConnected() {
    for (int i = 0; i < n; i++) visited[i] = 0;

    dfs(0);

    for (int i = 0; i < n; i++) {
        if (!visited[i]) return 0;
    }
    return 1;
}

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

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }

    if (isConnected()) {
        printf("The graph is connected.\n");
```

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

    return 0;
}
```

Output