VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SUSHANTH RAI (1BM24CS425)

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 **SUSHANTH RAI(1BM24CS425),** 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.

Ramya K M Assistant Professor Department of CSE, BMSCE	Dr. Jyothi S Nayak Professor &HOD Department of CSE, BMSCE

Index

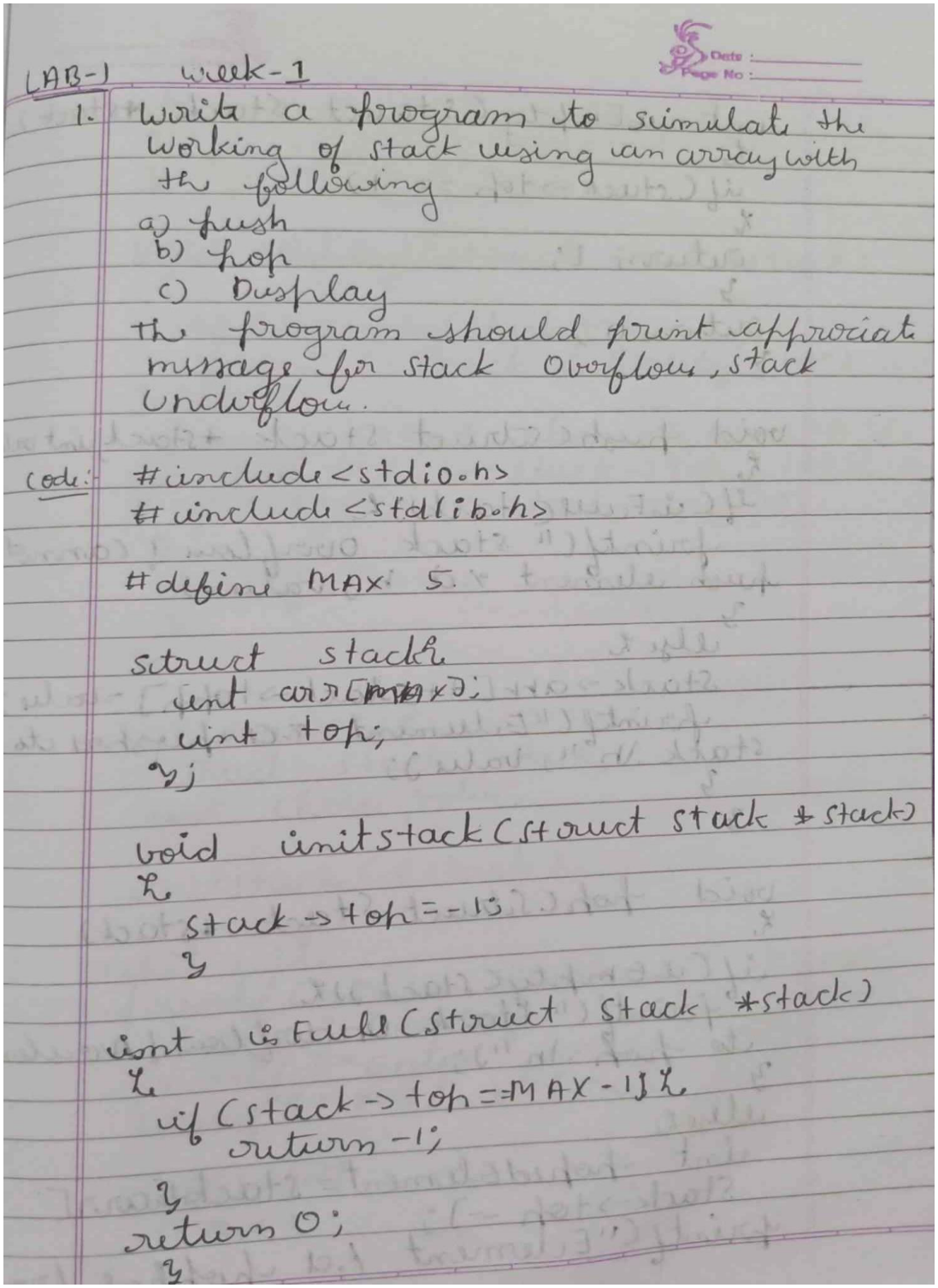
Sl. No.	Date	Experiment Title	Page No.
1	30/09/2024	Stack Implementation using arrays	1-8
2	07/10/2024	Infix to Postfix Conversion	9-14
3	15/10/2024	Queue implementation using arrays	15-22
4	21/10/2024	Circular Queue implementation using arrays	23-45
5	29/10/2024	Insertion operation in Singly linked list + Leetcode(Valid Paranthesis)	46-57
6	11/11/2024	Deletion operation in Singly linked list + Leetcode (Daily temperatures)	58-72
7	02/12/2024	Sorting,reversing,concatenating linked lists	73-86
8	02/12/2024	Stack implementation using linked list and Linear Queue implementation using linked list	87-102
9	16/12/2024	Insertion operation in Doubly linked list	103-112
10	23/12/2024	Binary Search Tree: Implementation and Traversal	113-121
11	23/12/2024	Graph Traversal using BFS and DFS method	122-127

Github Link: https://github.com/sushanthraiurwa/1BM24CS425-DS

LABORATORY 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

OBSERVATION :



LAB-1 week-1

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 message for stack Overflow, stack Underflow.

Code:

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

#define MAX 5

struct stack{
  int arr[MAX];
  int top;
};

void initstack(struct stack *stack)
{
  stack->top=-1;
}

int isFull(struct stack *stack)
{
  if(stack->top==MAX-1){
    return -1;
  }
  return 0;
}
```

```
int isEmpty(struct Stack *stack)
{
 if(stack->top == -1)
 {
 return 1;
 }
return 0;
}

void push(struct Stack *stack, int va
{
 if(isFull(stack)){
  printf("Stack Overflow! Cannot
push element %d\n", value);
 }
  else{
  stack->arr[++stack->top] = value
  printf("Element %d pushed to
stack.\n", value);
 }
}

void pop(struct Stack *stack)
{
 if(isEmpty(stack)){
  printf("Stack Underflow! No elem
  to pop.\n");
 }
 else{
  int poppedElement = stack->arr[
  stack->top--];
printf("Element %d popped from
```

```
stack \n", stack.arr[stack.top]);
}
}

void display(struct stack *stack)
{
  if (isEmpty(stack)){
    printf("Stack is Empty !\n");
}else{
  printf("stack elements are : \n");
for(int i=0; i<=stack->top; i++){
  printf("%d", stack->arr[i]);
}
  printf("\n");
}
}

int main(){
  struct stack stack;
  int choice, value;

  initstack(&stack);

while(1){
printf("stack operations : 1.push
2.pop 3. Display 4.exit");
  printf("Enter your choice :");
  scanf("%d", &choice);

switch(choice){
  case 1:
```

```
printf("Enter the elements to push");
scanf("%d", &value);
push(&stack, value);
break;

case 2:
pop(&stack);
break;

case 3:
display(&stack);
break;

case 4:
exit(0);

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

Output:-

Stack Operations
1. push 2. pop 3. Display 4. exit
Enter your choice 1
Enter the element to push: 10.
Element 10 pushed to stack.

Stack operations
1. push 2. pop 3. display 4. exit
Enter your choice 3
STACK element are:
10

STACK Operations:
1. push 2. pop 3. display 4. exit
Enter your choice 2.
Element 10. poped from the stack.

Stack operations
1. push 2. pop 3. display 4. exit
Enter your choice 3
Stack is Empty.

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");
        } }
}
```

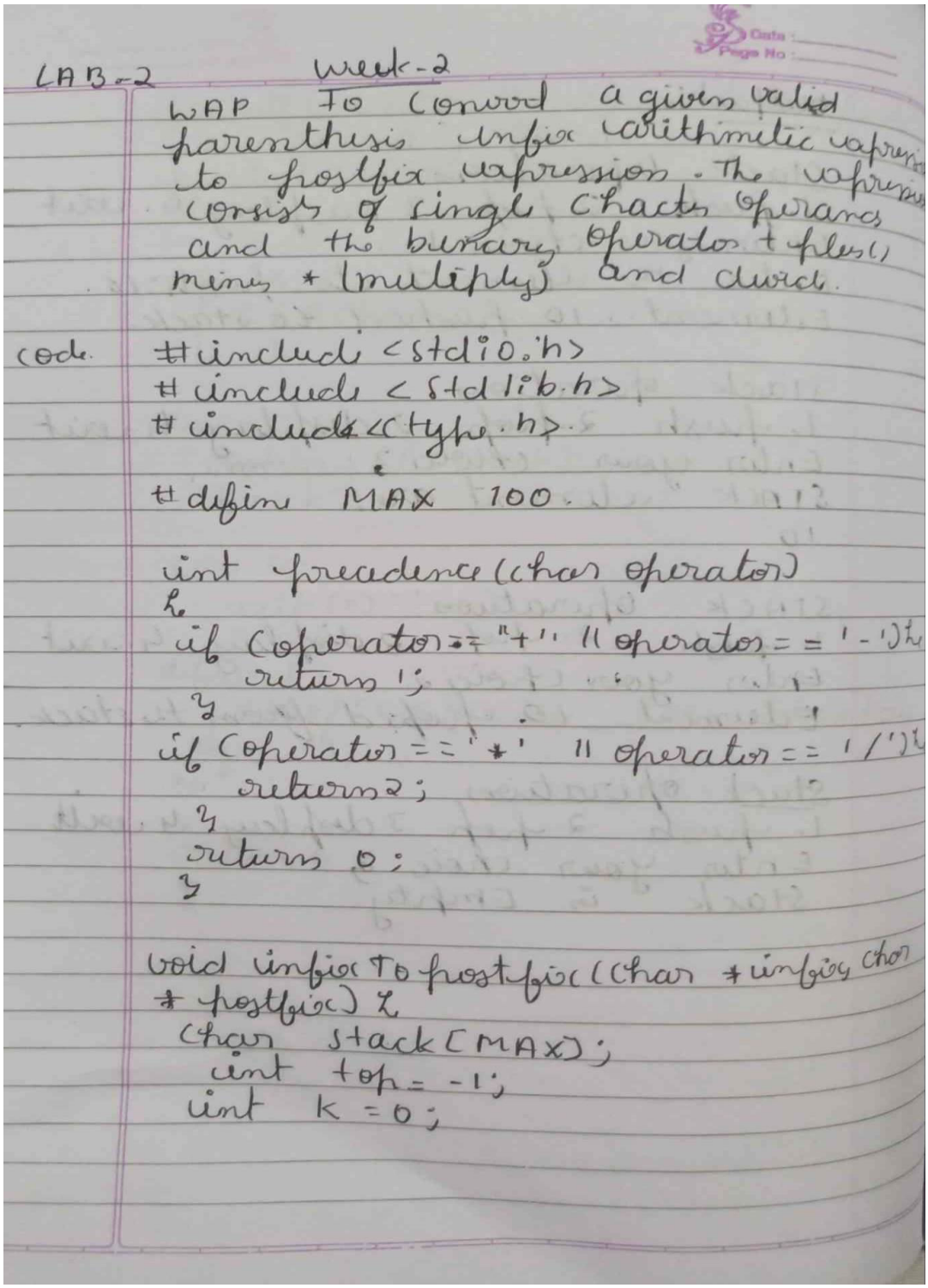
OUTPUT :

```
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.
```

LABORATORY PROGRAM – 2

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

OBSERVATION :



LAB-2 Week-2

WAP To Convert a given valid parenthesis infix arithmetic expression to postfix expression. The expression consists of single Chacter operands and the binary operator + (plus) minus * (multiply) and divide.

Code.

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

#define MAX 100

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

void infixToPostfix(char *infix, char
*postfix) {
 char stack[MAX];
 int top = -1;
 int k = 0;
```

```
for(int i=0; infix[i] != '\0'; i++){
    char token = infix[i];

    if(isalnum(token)){
        postfix[k++] = token;
    }
    else if(token == '('){
        stack[++top] = token;
    }
    else if(token == ')'){
        while(top != -1 && stop[top] != '('){
            postfix[k++] = stack[top--];
        }
        top--;
    }
    else if(token == '+' || token == '-' ||
    token == '*' || token == '/'){
        while(top != -1 && precedence(stack
        [top]) >= precedence(token)){
            postfix[k++] = stack[top--];
        }
        stack[++top] = token;
    }
}

while(top != -1){
    postfix[k++] = stack[top--];
}
postfix[k] = '\0';
}
```

```
int main(){
  char infix[MAX], postfix[MAX
  printf("Enter a valid infix expre
  fgets(infix, MAX, stdin);
  infix[strcspn(infix, "\n")] = '\
  infixTopostfix(infix, postfix);
  printf("Post expression: %s\n",
    postfix);
  return 0;
}
```

output:-

Enter a valid infix expression:
(a+b)*(c-d).

postfix expression: ab+cd-*

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
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) {
   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 ((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("Postfix expression: %s\n", result);
}
int main() {
    char exp[MAX];
    printf("Enter an infix expression: ");
    fgets(exp, MAX, stdin);
    exp[strcspn(exp, "\n")] = 0;
    infixToPostfix(exp);
    return 0;
}
```

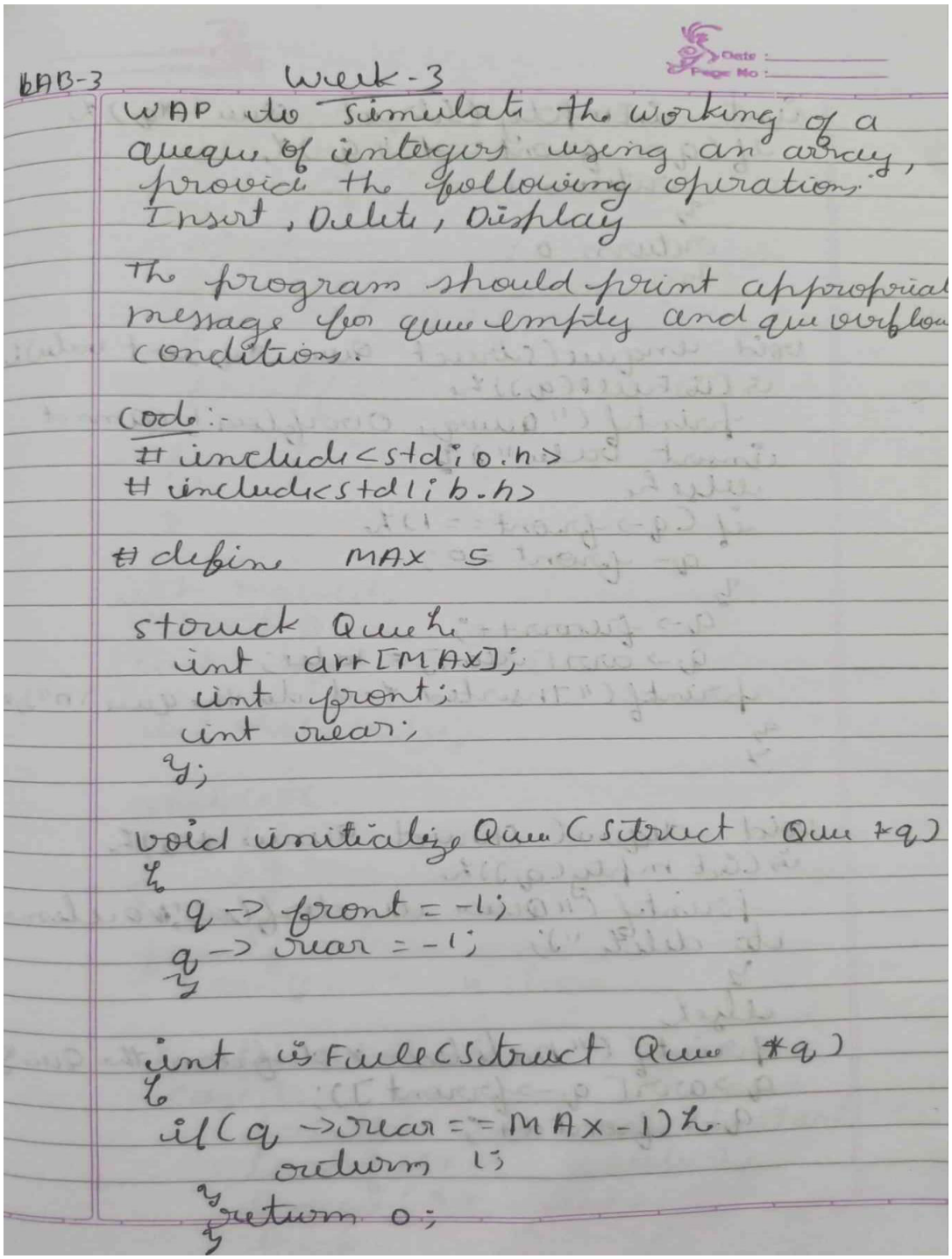
OUTPUT :

```
PS D:\3rd sem\DSA\lab programs> ./a.exe
Enter an infix expression: (a+b)*(c-d)
Postfix expression: ab+cd-*
PS D:\3rd sem\DSA\lab programs>
```

LABORATORY 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 messagesfor queue empty and queue overflow conditions

OBSERVATION :



LAB-3 Week-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 message for queue empty and queue overflow conditions.

Code:-

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

#define MAX 5

struct Queue
{
  int arr[MAX];
  int front;
  int rear;
};

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

int isFull(struct Queue *q)
{
  if(q->rear == MAX-1)
    return 1;
  return 0;
}
```

```
int isEmpty(struct Queue *q){
 if(q->front==-1 || q->front>q->rear){
    return 1;
  }
  return 0;
  }

void enqueue(struct Queue *q, int value){
 if(isFull(q)){
  printf("Queue Overflow! Cannot
insert value");}
 else{
 if(q->front==-1){
   q->front=0;
  }
   q->rear++;
   q->arr[rear]=value;
 printf("Inserted %d to the queue\n");
}
}

void dequeue(struct Queue *q){
 if(isEmpty(q)){
  printf("Queue underflow! No elem
to delete");
  }
  else{
 printf("Deleted %d from the Que
 q->arr[q->front]);
  q->front++;
```

```
void display(struct Queue *q){
  if(isEmpty(q)){
    printf("Queue is empty\n");
  }
  else{
  printf("Queue elements:");
  for(int i = q->front; i<=q->rear;i++
  {
  printf("%d", q->arr[i]);
  }
  printf("\n");
  }
}

int main(){
  struct Queue q;
  initialize Queue(&q);

  int choice, value;

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

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

```
case 2:
    dequeue(&q);
    break;

case 3:
    display(&q);
    break;

case 4:
    printf("Exiting program\n");
    exit(0);

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

OUTPUT:-

```
Queue Operation Menu:
1. Insert 2. Delete 3. display 4. exit
Enter your choice: 1
Enter value to insert:

Queue Operation Menu:
1. Insert 2. Delete 3. display 4. exit
Enter your choice 2.
Queue element: 10.
```

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int isFull(int rear) {
   if (rear == MAX - 1) {
      return 1;
   }
   return 0; }
int isEmpty(int front, int rear) {
   if (front == -1 || front > rear) {
      return 1;
   }
   return 0; }
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;
   }
   (*rear)++;
   queue[*rear] = value;
   printf("%d inserted into the queue\n", value);
}
```

```
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)++;
    if (*front > *rear) {
        *front = *rear = -1;
    }
}
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");
}
int main() {
    int queue[MAX];
    int front = -1, rear = -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;
}
```

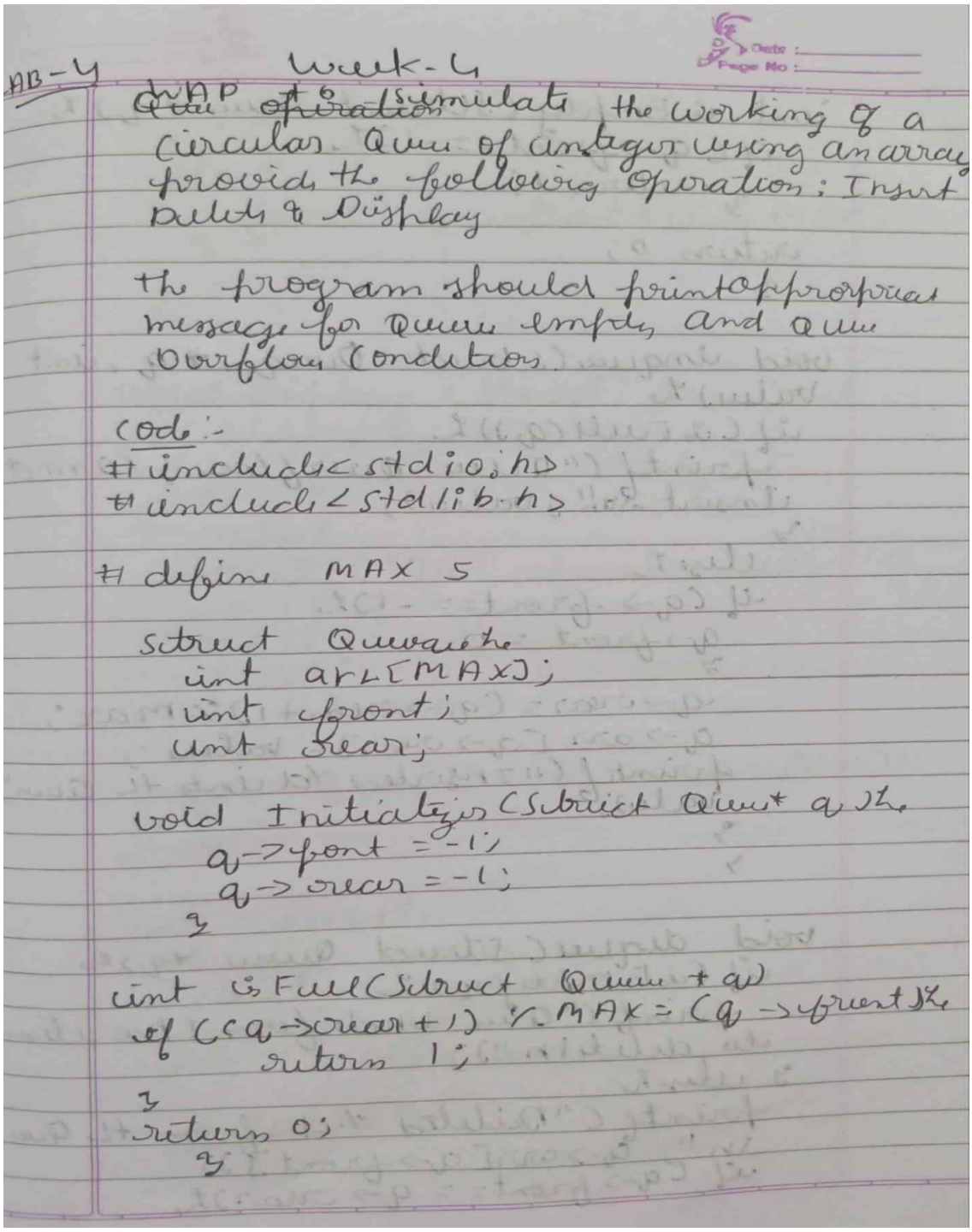
OUTPUT :

```
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
```

LABORATORY 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 messagesfor queue empty and queue overflow conditions

OBSERVATION :



AB-4 Week-4

WAP to simulate the working of a circular Queue of integer using an array provide the following operation: Insert Delete & Display

the program should print appropriate message for Queue empty and Queue Overflow Condition.

Code:-

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

#define MAX 5

struct Queue{
  int arr[MAX];
  int front;
  int rear;

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

int isFull(struct Queue *q)
if((q->rear+1) % MAX == (q->front)){
      return 1;
}
return 0;
}
```

```
int isEmpty(struct Queue *q){
  if(q->front==-1){
    return 1;
  }
  return 0;
}

void enqueue(struct Queue *q, int value){
  if(isFull(q)){
    printf("Queue Overflow! Cannot insert %d", value);
  }
  else{
    if(q->front==-1){
      q->front=0
    }
    q->rear=(q->rear+1)%MAX;
    q->arr[q->rear]=value;
    printf("Inserted %d into the Queue", value);
  }
}

void dequeue(struct Queue *q){
  if(isEmpty(q)){
    printf("Queue Underflow! No elements to delete\n");
  }else{
    printf("Deleted %d from the Queue\n", q->arr[q->front]);
    if(q->front==q->rear){
```

```
    q->front = q->rear = -1;
  }
  else
  q->front = (q->front + 1) % MAX;
  }
}

void display (struct Queue *q){
  if (isEmpty (q)){
  printf("Queue is Empty");
  }
  else{
  printf("Queue elements : ");
  int i = q->front;
  while (i != q->rear){
  printf("%d ", q->arr[i]);
  i = (i+1) % MAX;
  }
  printf("%d ", q->arr[q->rear]);
  }
}

int main(){
  struct Queue q;
  initialize Queue(&q);

  int choice, value;

  while(1){
```

```
printf("\nCircular queue operations\n");
printf("1. Insert 2. Delete 3. display
4. display 4. exit\n");
printf("Enter your choice :");
scanf("%d", &choice);

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

case 2:
dequeue(&q);
break;

case 3:
display(&q);
break;

case 4:
printf("Existing program");
exit(0);

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

Output:

Circular Queue Operation Menu.
1. Insert (Enqueue) 2. Delete Dequeue
3. display 4. exit.
Enter your choice 1
Enter value to insert: 10
Inserted 10 into the Queue.

Circular Queue operation Menu.
1. Insert (Enqueue) 2. Delete (Dequeue) 3. display
4. exit.
Enter your choice 3
Queue element: 10 ~~20~~

Circular Queue operation Menu
1. Insert (Enqueue) 2. Dequeue (Dequeue) 3. Display
4. exit.
Enter your choice 2
Deleted 10 from the Queue.

Circular Queue operations Menu.
1. Insert (Enqueue) 2. Dequeue 3. display
4. exit
Enter your choice 3
Queue is Empty.

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int isFull(int front, int rear) {
   if ((rear + 1) % MAX == front) {
      return 1;  }
   return 0;  }
int isEmpty(int front, int rear) {
   if (front == -1) {
      return 1;
   }
   return 0;
}
void insert(int queue[], int *front, int *rear, int value) {
   if (isFull(*front, *rear)) {
      printf("Queue Overflow! Cannot insert %d\n", value);
      return;
   }
   if (*front == -1) {
      *front = *rear = 0;
   } else {
      *rear = (*rear + 1) % MAX;
   }

   queue[*rear] = value;
```

```
    printf("%d inserted into the queue\n", value);
}

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 (*front == *rear) {
        *front = *rear = -1;
    } else {
        *front = (*front + 1) % MAX;
    }
}

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");
}

int main() {
    int queue[MAX];
    int front = -1, rear = -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;
}
```

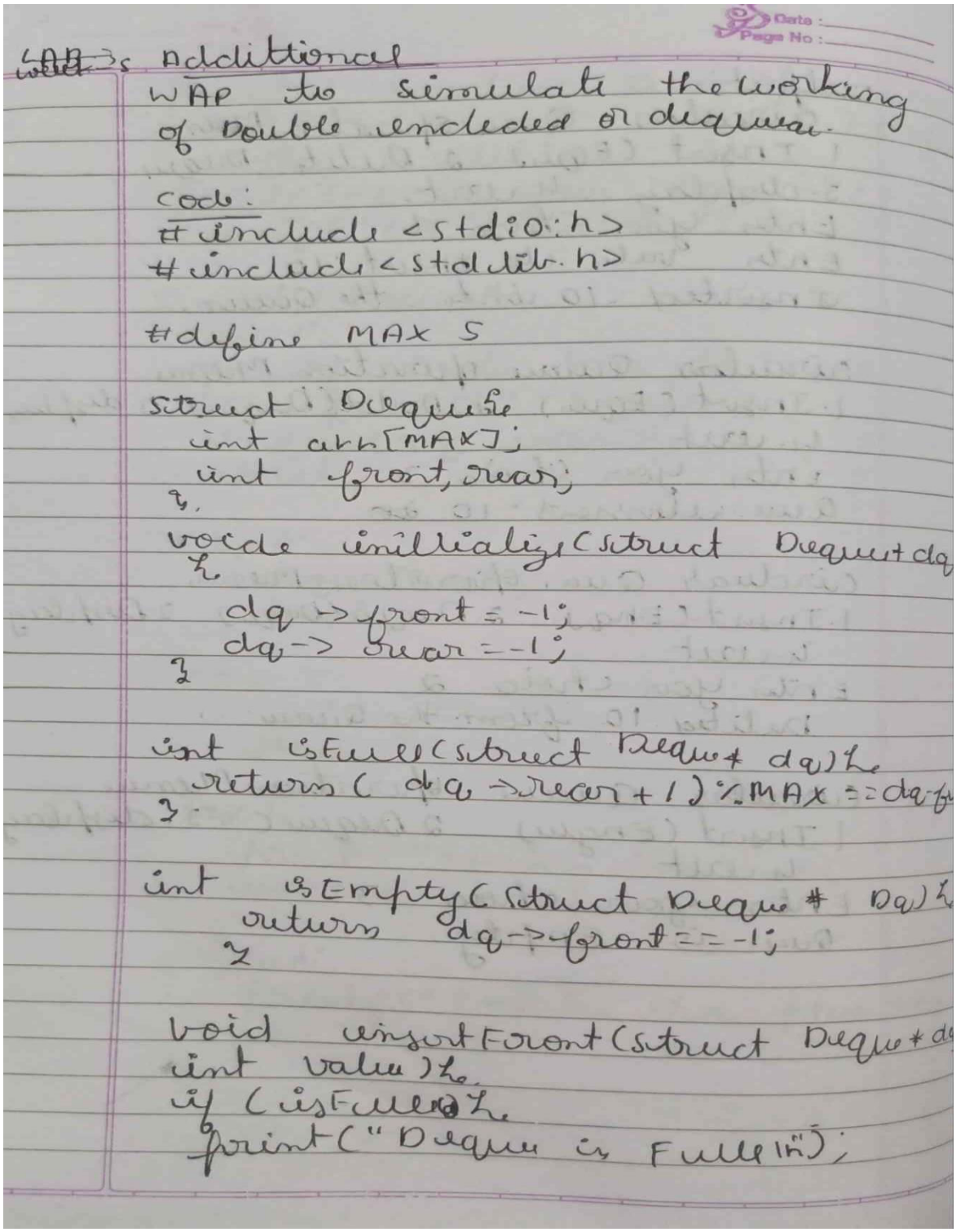
OUTPUT :

```
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

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 15

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

ADDITTIONAL :

WAP to simulate the working of Double ended queue or deque.

OBSERVATION :



Additional

WAP to simulate the working of Double ended or dequeue.

code:

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

#define MAX 5

struct Dequeue
  int arr[MAX];
  int front, rear;
};

voide initialize(struct Dequeue dq
{
   dq -> front = -1;
   dq -> rear = -1;
}

int isFull(struct Deque * dq){
  return ( dq -> rear + 1) % MAX == dq-fr
}

int isEmpty( struct Deque * Dq){
    return dq -> front == -1;
}

void insertFront(struct Deque * d
int value){
if (isFull){
 print("Deque is Full\n");
```

```
return;
}
if (dq->front == -1)
    dq->front = dq->rear = 0;
}
else
    dq->front = (dq->front - 1 + MAX) % MAX;
dq->arr[dq->front] = value;
}

void insertRear(struct Dequeue *dq, int value)
{
if(isFull(dq))
printf("The queue is Full");
return;
}
if(dq->front == -1)
    dq->front = dq->rear = 0;
}
else
    dq->rear = (dq->rear + 1) % MAX;
}
dq->arr[dq->rear] = value;
}

void deleteFront(struct Dequeue *dq)
{
if(isEmpty(dq))
printf("The Queue is Empty");
return;
}
```

```
int deletevalue = dq->arr[dq->f
if (dq->front == dq->rear){
    dq->front = dq->rear = -1;
}
else{
    dq->front = (dq->front+1)%M
}
printf("Deleted %d from the
front in", deletedvalue);
}

void deleteRear(struct Dequeue * dq
{
if (isEmpty(dq)){
printf("The Queue is Empty");
return;
}
int deletevalue = dq->arr[dq->re
if(dq->front == dq->rear){
    dq->front = dq->rear = -1;
}
 dq->rear = dq->(dq->rear-1+MAX)%
                                MAX
printf("Delete %d from the rear",
 deletedvalue);
}
```

```
void display(struct Dequeue *dq)
{
  if(isEmpty(dq)){
    printf("Dequeue is Empty");
    return;
  }
  printf("Dequeue elements:");
  int i = dq->front;
  while(i != dq->rear){
    printf("%d", dq->arr[i]);
    i = (i+1)%MAX;
  }
  printf("%d\n", dq->arr[dq->rear]);
}

int main(){
  struct Dequeue dq;
  initialize(&dq);

  int choice, value;

  while(1){
    printf("\nDequeue Operation Menu:\n");
    printf("1. Insert Front\n");
    printf("2. Insert Rear\n");
    printf("3. Delete Front\n");
    printf("4. Delete rear\n");
    printf("5. Display Dequeue\n");
    printf("6. Exit\n");
    printf("Enter your choice:");
```

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

switch(choice){
  case 1:
    printf("Enter the value to be insert
    at front : ");
    scanf("%d", &value);
    insertFront(&dq, value);
    break;

  case 2:
    printf("Enter the value to be insert
    rear : ");
    scanf("%d", &value);
    insertRear(&dq, value);
    break;

  case 3:
    deleteFront(&dq);
    break;

  case 4:
    deleteRear(&dq);
    break;

  case 5:
    display(&dq);
    break;

  case 6:
    printf("Exiting");
    exit(0);
    break;
```

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

output:-

```
Deque Operation Menu.
1. Insert front  2. Insert Rear
3. Delete front  4. Delete Rear
5. Display Deque  6. Exit
Enter your choice 1
Enter the value ito to insert At front1
```

```
Deque Operation Menu.
1. Insert front  2. Insert Rear.
3. Delete front  4. Delete Rear.
5. Display Deque  6. Exit
Enter your choice 2
Enter the value to insert At end 20
```

```
Deque Operation Menu
1. Insert front  2. Insert Rear.
3. Delete front  4. Delete Rear
5. Display Deque  6. Exit
Enter your choice 3.
Deleted 10 from the front.
```

Deque operation Menu
1. Insert front 2. Insert Rear
3. Delete front 4. Delete Rear
5. Display Queue 6. Exit.
Enter your choice 4
Deletion do from the rear.

Deque operation Menu.
1. Insert front 2. Insert Rear
3. Delete front 4. Delete Rear.
5. Display Queue 6. Exit
Enter your choice 5.
Empty queue!

CODE :

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
struct Deque {
   int arr[MAX];
   int front, rear;
};
void initialize(struct Deque* dq) {
   dq->front = -1;
   dq->rear = -1;
}
int isFull(struct Deque* dq) {
   return (dq->rear + 1) % MAX == dq->front;
}

int isEmpty(struct Deque* dq) {
   return dq->front == -1;
}

void insertFront(struct Deque* dq, int value) {
   if (isFull(dq)) {
      printf("Que is full\n");
      return;
   }
```

```
    if (dq->front == -1) {
        dq->front = dq->rear = 0;
    } else {
        dq->front = (dq->front - 1 + MAX) % MAX;
    }
    dq->arr[dq->front] = value;
}
void insertRear(struct Deque* dq, int value) {
    if (isFull(dq)) {
        printf("Que is full.\n");
        return;
    }
    if (dq->front == -1) {
        dq->front = dq->rear = 0;
    } else {
        dq->rear = (dq->rear + 1) % MAX;
    }
    dq->arr[dq->rear] = value;
}
void deleteFront(struct Deque* dq) {
    if (isEmpty(dq)) {
        printf("Que is Empty.\n");
        return;
    }
    int deletedValue = dq->arr[dq->front];
    if (dq->front == dq->rear) {
```

```
        dq->front = dq->rear = -1;
    } else {
        dq->front = (dq->front + 1) % MAX;
    }
    printf("Deleted %d from the front.\n", deletedValue);
}
void deleteRear(struct Deque* dq) {
    if (isEmpty(dq)) {
        printf("Que is Empty.\n");
        return;
    }
    int deletedValue = dq->arr[dq->rear];
    if (dq->front == dq->rear) {
        dq->front = dq->rear = -1;
    } else {
        dq->rear = (dq->rear - 1 + MAX) % MAX;
    }
    printf("Deleted %d from the rear.\n", deletedValue);
}
void display(struct Deque* dq) {
    if (isEmpty(dq)) {
        printf("Deque is empty!\n");
        return;
    }
    printf("Deque elements: ");
    int i = dq->front;
```

```
    while (i != dq->rear) {
        printf("%d ", dq->arr[i]);
        i = (i + 1) % MAX;
    }
    printf("%d\n", dq->arr[dq->rear]);
}
int main() {
    struct Deque dq;
    initialize(&dq);
    int choice, value;
    while(1) {
        printf("\nDeque Operations Menu:\n");
        printf("1. Insert Front ");
        printf("2. Insert Rear ");
        printf("3. Delete Front ");
        printf("4. Delete Rear ");
        printf("5. Display Deque ");
        printf("6. Exit ");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert at front: ");
                scanf("%d", &value);
                insertFront(&dq, value);
                break;
```

```
            case 2:
                printf("Enter value to insert at rear: ");
                scanf("%d", &value);
                insertRear(&dq, value);
                break;
            case 3:
                deleteFront(&dq);
                break;
            case 4:
                deleteRear(&dq);
                break;
            case 5:
                display(&dq);
                break;
            case 6:
                printf("Exiting...\n");
                exit(1);
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT :

```
PS D:\3rd sem\DSA\CIE2> ./a.exe

Deque Operations Menu:
1. Insert Front 2. Insert Rear 3. Delete Front 4. Delete Rear 5. Display Deque 6. Exit
Enter your choice: 1
Enter value to insert at front: 10

Deque Operations Menu:
1. Insert Front 2. Insert Rear 3. Delete Front 4. Delete Rear 5. Display Deque 6. Exit
Enter your choice: 2
Enter value to insert at rear: 20

Deque Operations Menu:
1. Insert Front 2. Insert Rear 3. Delete Front 4. Delete Rear 5. Display Deque 6. Exit
Enter your choice: 3
Deleted 10 from the front.

Deque Operations Menu:
1. Insert Front 2. Insert Rear 3. Delete Front 4. Delete Rear 5. Display Deque 6. Exit
Enter your choice: 4
Deleted 20 from the rear.

Deque Operations Menu:
1. Insert Front 2. Insert Rear 3. Delete Front 4. Delete Rear 5. Display Deque 6. Exit
Enter your choice: 5
Deque is empty!

Deque Operations Menu:
1. Insert Front 2. Insert Rear 3. Delete Front 4. Delete Rear 5. Display Deque 6. Exit
Enter your choice: 6
Exiting...
PS D:\3rd sem\DSA\CIE2>
```

LABORATORY PROGRAM – 5

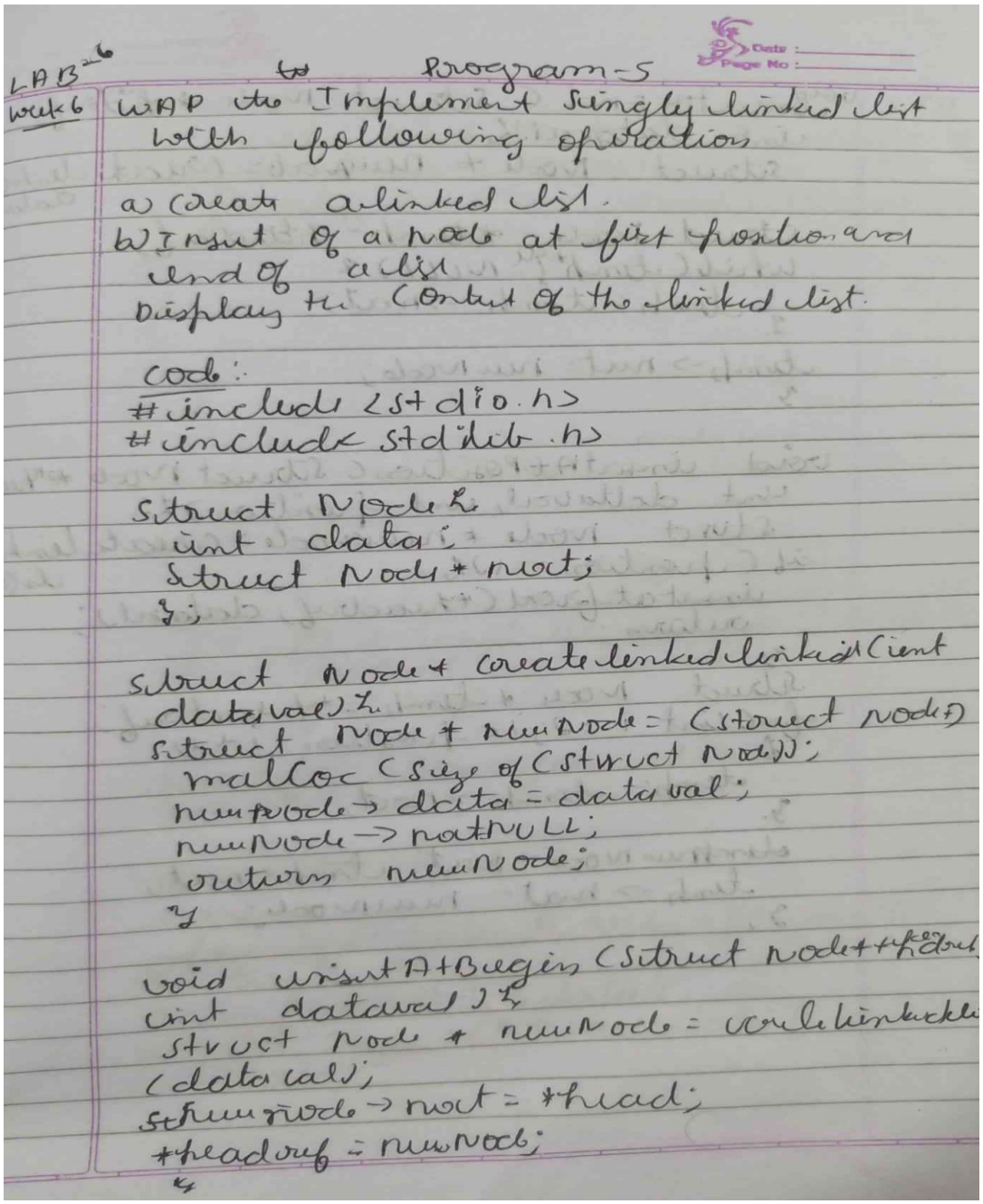
5a)WAP to Implement Singly Linked List with following operations

a) Createalinkedlist.

b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.

5b) Leetcode problem no.20 (Valid parantheses)

OBSERVATION :



```
void insertAtEnd ( struct node * headref,
  int dataval){
   struct Node * newNode = (create
                                  da
   struct Node *temp = *headref;
   while(temp->next != NULL){
      temp = temp->next;
   }
   temp->next = newNode;
}

void insertAtPosition( struct Node **h
  int dataval, int position ){
   Struct Node * newNode = (createLin
                                  list
  if (position == 1){
     insertatfront (*headref, dataval);
     return;
  }
  struct Node * temp = *headref;
  for(int i = 2; i < position ; i++)
  {
     temp = temp->next;
  }
  newNode->next = 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("\n");
}

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

 while(1){
 printf("1. Insert At Begining:");
 printf("2. Insert At the end:");
 printf("3. Insert At specific position");
 printf("4. Display the list");
 printf("5. Exit\n");

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

 switch(choice){
 case 1:
  printf("-Enter the value to insertAt
the Beging");
 scanf("%d", &data);
 insertAtBegin(&head, data);
  break;
```

```
case 2:
    printf("Enter the value to insert
the end");
    scanf("%d", &data);
insertAtEnd(head, data);
break;

case 3:
printf("Enter the value to Insert");
scanf("%d", &data);
printf("Enter the position to insert At");
scanf("%d", &position);
insertAtPosition(&head, data, position);
break;

case 4:
    display(head);
    break;

case 5:
    printf("Exiting \n");
    break;
    exit(1);

default:
    printf("Invalid choice plea try ag
    }
  }
}
return 0;
}
```

Output:

Menu
1. Insut At Beging 2. Insut At endEnd
3. Insut At Specific position 4. Display so
Enter your choice : 1
Enter the to insut At Beging 10.

Menu
1. InsertAt Beging 2. InsertAtend
3. Insert At Specific position 4. Display so
Enter your choice 2
Enter the value to insut at end: 20.

Menu:
1. InsertAtBegining 2. InsrtAtEnd.
3. Insd At Specific portion 4. Display soc
Enter your choice 3
Enter the valu to be insated 30
Enter the position 3

Menu
1. InsutAt Begining 2. InsertAtEnd
3. Insut At specific position 4 Display so
Enter your choice 4.

10 → 20 → 30 →

CODE :

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

struct Node {
    int data;
    struct Node* next;
};
void createList(struct Node** head) {
    *head = NULL;
}
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);
}
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);
        free(newNode);
    } else {
        newNode->next = temp->next;
        temp->next = newNode;
        printf("Node with value %d inserted at position %d.\n", value,
position);
    }
}

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);
}

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");
}
```

```
int main() {
    struct Node* head;
    createList(&head);
    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
}
```

LABORATORY PROGRAM – 6

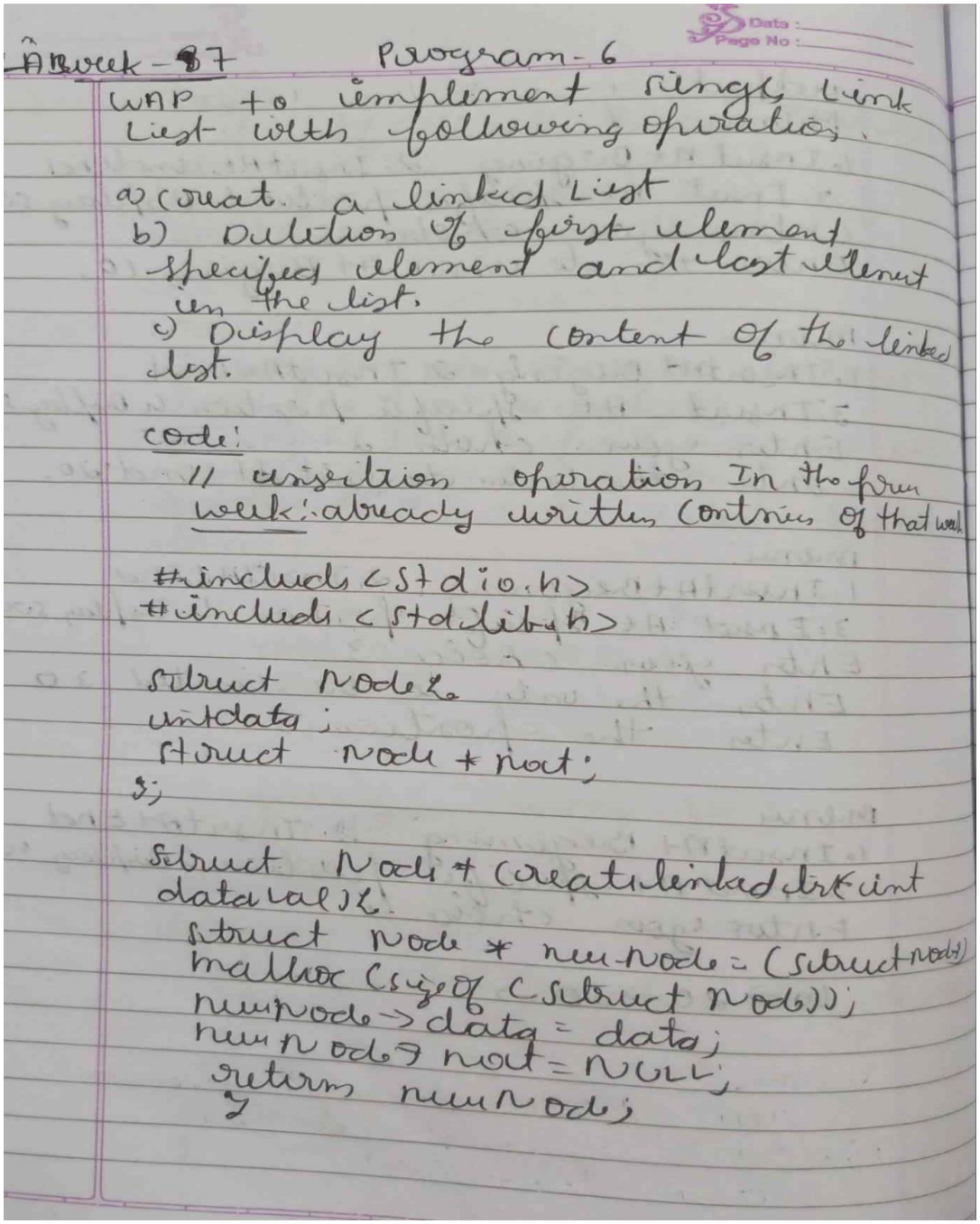
WAP to Implement Singly Linked List with following operations

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

OBSERVATION :



Week - 7 Program - 6

WAP to implement singly Link List with following operations.

a) creat a linked List
b) Deletion of first element specified element and last element in the list.
c) Display the content of the linked list.

code:

// insertion operation In the prev week: already written contents of that week

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

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

struct Node *createlinkedlist(int
datavalue){
  struct Node * newNode = (struct Node)
  malloc (sizeof (struct Node));
  newnode->data = data;
  newNode->next = NULL;
  return newNode;
}
```

```
void deleteAtfront(struct node *
headref){
struct node *temp = *headref;
*headref = temp->next;
free(temp);
}

void deleteAtEnd(struct node *
headref)
struct node *temp = headref;
while(temp->next != NULL && temp->
next->next != NULL){
temp = temp->next;
}
struct node *last = temp->next;
temp->next = NULL;
free(last);
}

void deleteAtposition(struct node
**headref,int position){
if(*headref == NULL){
printf("List is Empty");
outern;
else if(position == 1){
delete Atfront(headref);
}
else{
struct node *temp = *headref;
for(int i=2; i<position; i++){
temp = temp->next;
}
```

```
Struct Node *del = temp->next;
temp->next = del->next;
free(del);
}
}

void display (Struct Node *hea
{
struct node *temp = headref;

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

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

while(1)
printf("Menu\n");
printf("1. InsertAt Beginig");
printf("2. InsertAt End");
printf("3. InsertAt the position");
printf("4. Data at the front");
```

```
printf("5. delete At the end");
printf("6. Delete at the position\n");
printf("7. Display the list\n");
printf("8 Exit\n");

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

switch(choice){

case 1:
  printf("Enter the value to insert
At the Beginig:");
scanf("%d",&data);
Insert At Begin(&head, data);
break;

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

case 3:
  printf("Enter the value to insert");
  scanf("%d", &data);
  printf("Enter the position to insert
  scanf("%d", &position);
  insert At position(&head, data,
  position);
  break;
```

```
case 4:
    deleteAtfront(&head);
    break;

case 5:
    deleteAtEnd(head);
    break;

case 6:
    printf("Enter the position to
    delete at")
    scanf("%d", &position);
    deleteAtposition(&head, posi
    break;

case 7:
    display(head);
    break;

case 8:
    printf("Exiting");
    break;
    exit(1);

default:
    printf(Invalid Choice Pls try
    Again in");
}
}
```

Output :-

Menu:

1. Insert At beginning 2. Insert At end
3. Insert At position 4. Delete At pos
5. Delete At end 6. Delet at pos
7. Display the list. 8. Exit

Enter your choice:1
Enter the value to insert at the begin:1

Enter your choice : 2
Enter the value to insert At end : 2

Enter your choice:3
Enter the value 30
Enter the position 3

Enter your choice : 7
10 → 20 → 30 → NULL

Enter your choice : 4

Enter your choice : 5

Enter your choice : 7
20 → NULL

Enter your choice : 6
Enter the position to delet : 1

Enter your choice : 7
List is Empty.

CODE :

```
 #include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
void createList(struct Node** head) {
    *head = NULL;
}
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");
}
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 (temp != NULL && temp->data == value) {
        *head = temp->next;
        free(temp);
        printf("Element %d deleted successfully.\n", value);
        return;
    }
    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Element %d not found in the list.\n", value);
        return;
    }
    prev->next = temp->next;
    free(temp);
    printf("Element %d deleted successfully.\n", value);
}
void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty. No element to delete.\n");
        return;
    }
    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");
}

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");
}

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);
}

int main() {
    struct Node* head;

    createList(&head);
    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);
```

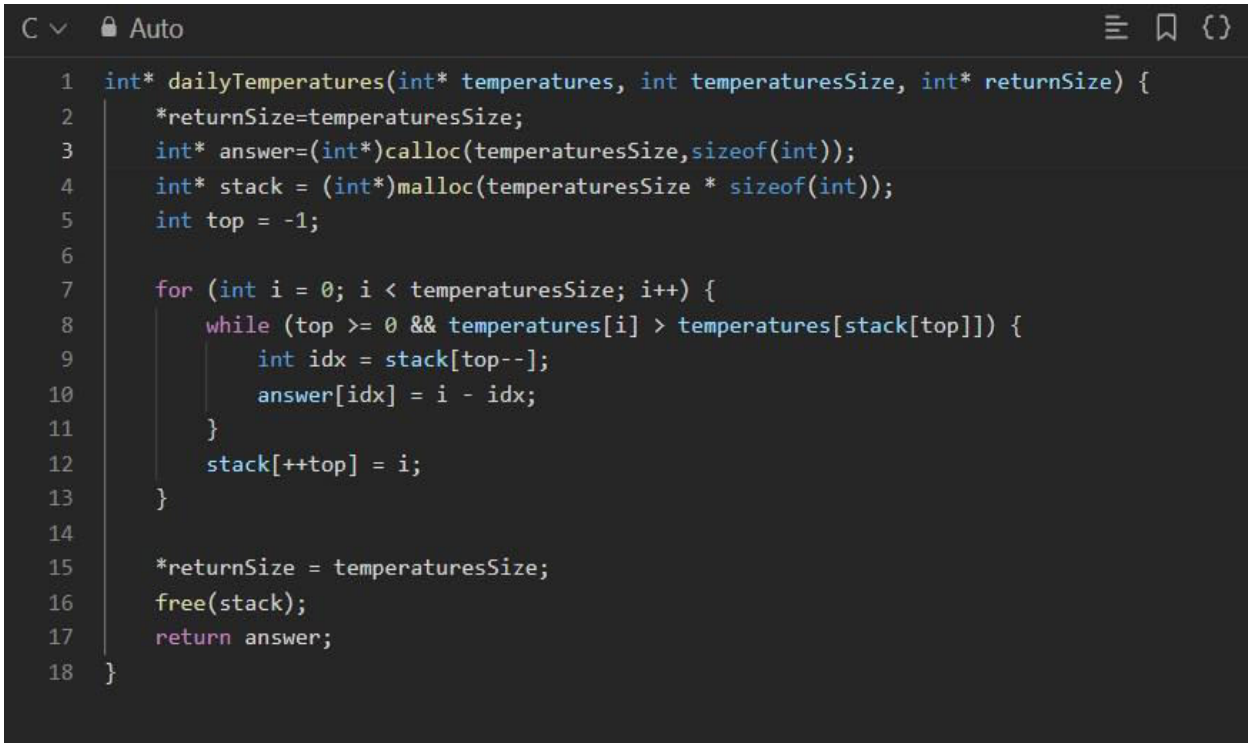
```
            break;

        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 :

```
Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 5
Linked List: 5 -> 18 -> 1478 -> NULL

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 3
Enter the value to delete: 1475
Element 1475 not found in the list.

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 6
PS C:\Users\Shashank U\Desktop\C>
```

```
Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 5
Linked List: 12 -> 5 -> 18 -> 1478 -> 14

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 2
First element deleted successfully.

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 3
Enter the value to delete: 14
Element 14 deleted successfully.

Singly Linked List Operations:
1. Insert at End
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display the list
6. Exit
Enter your choice: 4
Last element deleted successfully.
```

Leetcode problem(Daily Temperatures):

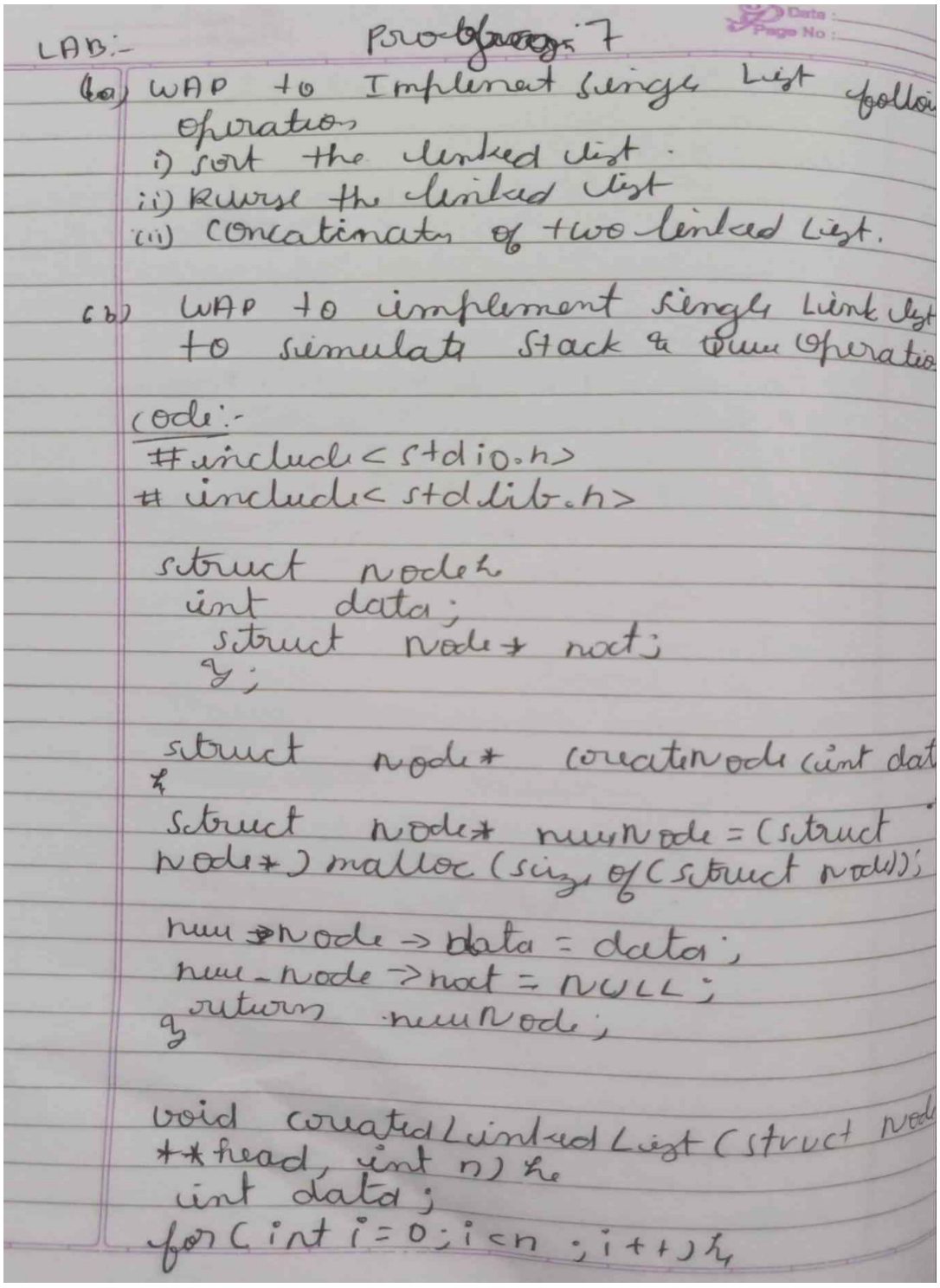


```c
int* dailyTemperatures(int* temperatures, int temperaturesSize, int* returnSize) {
    *returnSize=temperaturesSize;
    int* answer=(int*)calloc(temperaturesSize,sizeof(int));
    int* stack = (int*)malloc(temperaturesSize * sizeof(int));
    int top = -1;

    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;
}
```

LABORATORY PROGRAM – 7

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

OBSERVATION :



LAB:- Program 7

(a) WAP to Implement Single List follo
operations
i) sort the linked list.
ii) Rurse the linked list
(ii) Concatinate of two linked List.

(b) WAP to implement Single Link list
to simulate Stack & Queue Operatio

Code:-

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

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

struct node* creatNode(int dat
{
 struct node* newNode=(struct
node*)malloc(size of(struct node));

 newNode->data=data;
 new_node->next=NULL;
 return newNode;
}

void creatLinkedList(struct node
**head, int n){
 int data;
for(int i=0;i<n;i++){
```

```
printf("Enter data for node %d : ",
  i+1 );
scanf("%d", &data);
struct Node *newNode = (creatNode
 (data);
if (*head == NULL) {
    *head = newNode;
 }else{
   struct Node* temp = *head;
   while(temp -> next != NULL) {
    temp = temp -> next
   }
  temp -> next = newNode;
  }
}
}

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

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

void reverseList(struct node** head)
{
 struct node *prev=NULL,
 *current = *head,
 *next=NULL;

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

```
*head = prev;
}

void concatenateList(struct Node **hea
 struct Node *head2){

 if(*head1 = NULL)
 {
  *head1 = head2;
   return;
 }
 struct node* temp = *head1;
 while(temp -> next != NULL){
   temp = temp -> next;
  }
  temp -> next = head2;
}

~~while~~
int main(){

 struct Node*  head1 = NULL;
 struct Node*  head2 = NULL;

 int n1, n2, ~~choice~~, ~~data~~;

 printf("Enter the number of Nodes for
  the first linked list");
 scanf("%d", &n1);
 createLinkedList(&head1, n1);
```

```
printf("Linked list created \n");

printf("Enter the number of nodes of the
second linked list : ");
scanf("%d", &n2);
printf("Linked list 2 created. \n");

while(1){
printf("Menu \n");
printf("1. display linked list \n");
printf("2. sort the linked list \n");
printf("3. Reverse the linked list \n");
printf("4. Concatage Two linked list \n");
printf("5. exit \n");
printf("Enter your choice : ");
scanf("%d", &choice);

switch(choice){
case 1:
printf("Display Linked list : ");
display(head1);
break;

case 2:
if(head1 != NULL){
sortlist(&head1);
printf("Linked list sorted");
else{
printf("List is Empty,
cannot sort. \n");
}
break;
```

```
case 3:
 if (head1 != NULL) {
 rev(&head1);
 printf ("Linked list reversed \n");
}
else
 printf ("List is Empty, cannot
 reverse. \n");
}
break;

case 4:
   concatenate (&head1, head2);
 printf ("Linked list 2 concated to
linked list 1. \n");
 break;

case 5:
  printf ("Exiting program \n");
exit(1);

default;
   printf ("Invalid Choice, please try
Again \n");
  }
  }
return 0;
}
```

Output

```
Enter the number of Nodes for the first
linked list : 3.
Enter data for Node 1 : 10
Enter data for Node 2 : 30
Enter data for Node 3 : 50

Linked list 1 created

Enter the Number of Nodes for the
Second linked list : 3
Enter data for Node 1 : 20
Enter data for Node 2. 40
Enter data for Node 3 : 60
Linked list 2 created.

Menu:
1. Display linked list
2. Sort the linked list
3. Reverse the linked list
4. Concatenate two linked list
5. Exit
Enter your choice 2.
Linked list 1 sorted

Enter your choice 3.
Linked list 1 reversed.
```

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 createLinkedList(struct Node** head, int n) {
    int data;
    struct Node* temp;

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

        struct Node* newNode = createNode(data);

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

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

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

void sortList(struct Node* head) {
    if (head == 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 reverse(struct Node** head) {
    struct Node *prev = NULL, *current = *head, *next = NULL;

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

    *head = prev;
}
```

```
void concatenate(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* head1 = NULL;
    struct Node* head2 = NULL;
    int choice, n1, n2;

    printf("Enter the number of nodes for the first linked list: ");
    scanf("%d", &n1);
    createLinkedList(&head1, n1);
    printf("Linked List 1 created.\n");

    printf("Enter the number of nodes for the second linked list: ");
```

```
scanf("%d", &n2);
createLinkedList(&head2, n2);
printf("Linked List 2 created.\n");

while (1) {
  printf("\nMenu:\n");
printf("1. Display Linked List\n");
printf("2. Sort Linked List\n");
printf("3. Reverse Linked List\n");
printf("4. Concatenate Two Linked Lists\n");
printf("5. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

   switch (choice) {
      case 1:
         printf("Linked List 1:\n");
         display(head1);
         break;

      case 2:
         if (head1 != NULL) {
            sortList(head1);
            printf("Linked List 1 sorted.\n");
         } else {
            printf("List 1 is empty, cannot sort.\n");
```

```
            }
            break;
        case 3:
            if (head1 != NULL) {
                reverse(&head1);
                printf("Linked List 1 reversed.\n");
            } else {
                printf("List 1 is empty, cannot reverse.\n");
            }
            break;

        case 4:
            concatenate(&head1, head2);
            printf("Linked List 2 concatenated to Linked List 1.\n");
            break;

        case 5:
            printf("Exiting program.\n");
            exit(1);
        default:
            printf("Invalid choice. Please try again.\n");
      }
   }

   return 0;
}
```

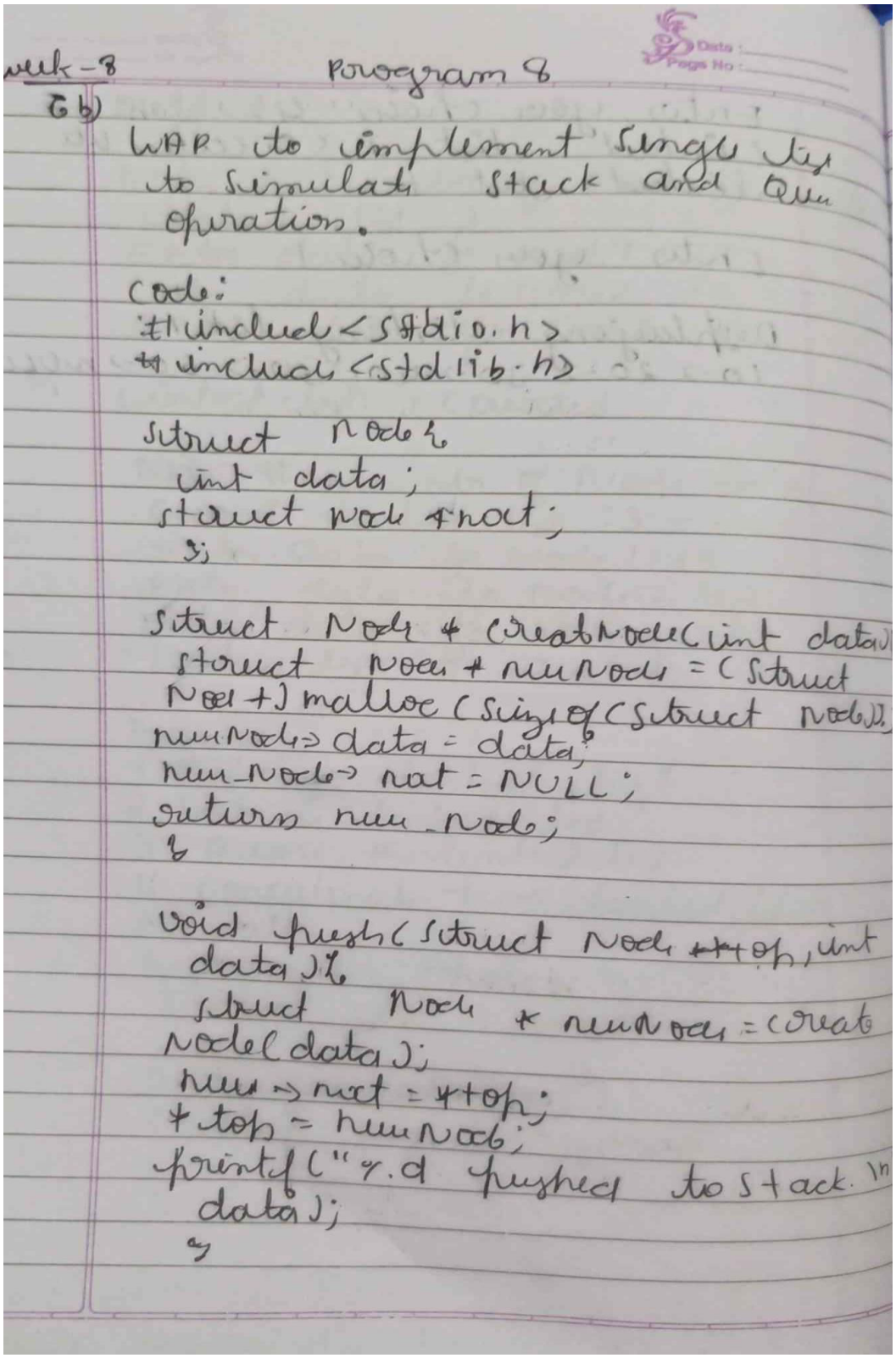
OUTPUT :

```
Enter the number of nodes for the first linked list: 3
Enter data for node 1: 10
Enter data for node 2: 30
Enter data for node 3: 50
Linked List 1 created.
Enter the number of nodes for the second linked list: 3
Enter data for node 1: 20
Enter data for node 2: 40
Enter data for node 3: 60
Linked List 2 created.

Menu:
1. Display Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Two Linked Lists
5. Exit
Enter your choice: 2
Linked List 1 sorted.

Menu:
1. Display Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Two Linked Lists
5. Exit
Enter your choice: 3
Linked List 1 reversed.

Menu:
1. Display Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Two Linked Lists
5. Exit
Enter your choice: 4
Linked List 2 concatenated to Linked List 1.

Menu:
1. Display Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Two Linked Lists
5. Exit
Enter your choice: 1
Linked List 1:
50 -> 30 -> 10 -> 20 -> 40 -> 60 -> NULL
```

LABORATORY PROGRAM – 8

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

OBSERVATION :



week-8 Program 8

6b)

WAP to implement Single list to simulate stack and Queue operation.

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

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

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

void push(struct Node **top,int
  data){
  struct Node *newNode=(creat
NodeCdata);
new->next=*top;
*top=newNode;
printf("%d pushed to stack.\n
  data);
}
```

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

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

void display stack(struct Node *top)
  if (top == NULL){
  printf("Stack is Empty");
    return;
  }
  struct Node *temp = top;
    printf("stack : ");
  while (temp != NULL){
    printf("%d", temp->data);
    temp = temp->next;
  }
```

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

void enqueue(struct node *front,
 struct Node *rear, int data)
{
struct node * newNode = (creatNode
                          data);

if (*rear == NULL) {
 *front = *rear = newNode;
 printf("%d Enqued to queue\n",
   data);
 return;
}
(*rear) -> next = newNode;
*rear = newNode;
printf("%d Enqueued to queue\n",
 data);
}

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

```
void peekQueue(struct node *front)
{
if (front == NULL)
{
printf("Queue is Empty\n");
return;
}
printf("Front element %d \n",
front -> data);
}

void displayQueue(struct node *
front)
{
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("\n");
}

int main()
{

struct Node * stackTop = NULL;
Struct Node * queueFront = NULL;
Struct Node * queueRear = NULL;

int choice, data;
```

```
while(1){
printf("Choose an operation");
printf("1. stack : push\n");
printf("2. stack : pop\n");
printf("3 Stack : peek\n");
printf("4 Stack : Display");
printf("5 Queue : Enqueue");
printf("6 Queue : Dequeue");
printf("4 Queue : peek");
printf("8 Queue : Display");
printf("9 : exit\n");

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

switch(choice){

case 1:
    printf("Enter the value to push to stack");
scanf("%d", &data);
push(&stacktop, data);
break;

case 2:
    pop(&stacktop);
    break;

case 3:
    peek(stacktop);
    break;
```

```
case 4:
    display stack (stacktop);
    break;

case 5:
    printf ("Enter the value to
    push");
    scanf ("%d", &data);
    enqueue (&qfront, &qrear,
    data);
    break;

case 6:
    dequeue (&qufront);
    break;

case 7:
    pushQueue (&qufront);
    break;

case 8:
    displayQueue (qufront);
    break;

case 9:
    printf ("Exiting the program\n");
    exit();

default:
    printf ("Invalid choice try Again");
}
```

Output:-

Choose an operation
1. Stack : Push
2. Stack : Pop
3 Stack : Peek
4 Stack : Display
5 Stack : Enqueue
6 Queue : Dequeue
7 Queue : Peek
8 Queue : Display
9 exit
Enter your choice : 1
Enter the value to push to Stack 10
10 pushed to stack

Enter your choice 1
Enter the value to push to Stack 20
20 pushed to stack

Enter your choice : 2
20 popped from stack

Enter your choice : 3
Top element : 10

Enter your choice : 4
Stack 10

Enter your choice: 8
Queue is Empty

Enter your choice 5
Enter the value enqueue to enqueue
10 enqueue to queue

Enter your choice 5
Enter the value enqueue to enqueue 2
20 enqueued to queue

Enter your choice: 6
10 dequeue from queue

Enter your choice: 7
Foreront element: 20

Enter your choice: 8
Queue: 20.

CODE :

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

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

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


void push(struct Node** top, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *top;
   *top = newNode;
   printf("%d pushed to stack.\n", data);
}

void pop(struct Node** top) {
   if (*top == NULL) {
      printf("Stack is empty.\n");
      return;
```

```
    }
    struct Node* temp = *top;
    *top = (*top)->next;
    printf("%d popped from stack.\n", temp->data);
    free(temp);
}

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

void displayStack(struct Node* top) {
    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("\n");
```

```
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        printf("%d enqueued to queue.\n", data);
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
    printf("%d enqueued to queue.\n", data);
}

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

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

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

void displayQueue(struct Node* front) {
    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("\n");
}

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

    int choice, data;
```

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

    switch (choice) {
        case 1:
            printf("Enter value to push to stack: ");
            scanf("%d", &data);
            push(&stackTop, data);
            break;
        case 2:
            pop(&stackTop);
            break;
        case 3:
            peek(stackTop);
            break;
        case 4:
```

```
            displayStack(stackTop);
            break;
        case 5:
            printf("Enter value to enqueue to queue: ");
            scanf("%d", &data);
            enqueue(&queueFront, &queueRear, data);
            break;
        case 6:
            dequeue(&queueFront);
            break;
        case 7:
            peekQueue(queueFront);
            break;
        case 8:
            displayQueue(queueFront);
            break;
        case 9:
            printf("Exiting the program.\n");
            exit(1);
        default:
            printf("Invalid choice. Try again.\n");
    }

  }
  return 0;
}
```

OUTPUT :

```
Choose an operation:
1. Stack: Push
2. Stack: Pop
3. Stack: Peek
4. Stack: Display
5. Queue: Enqueue
6. Queue: Dequeue
7. Queue: Peek
8. Queue: Display
9. Exit
Enter your choice: 1
Enter value to push to stack: 10
10 pushed to stack.

Choose an operation:
1. Stack: Push
2. Stack: Pop
3. Stack: Peek
4. Stack: Display
5. Queue: Enqueue
6. Queue: Dequeue
7. Queue: Peek
8. Queue: Display
9. Exit
Enter your choice: 1
Enter value to push to stack: 20
20 pushed to stack.

Choose an operation:
1. Stack: Push
2. Stack: Pop
3. Stack: Peek
4. Stack: Display
5. Queue: Enqueue
6. Queue: Dequeue
7. Queue: Peek
8. Queue: Display
9. Exit
Enter your choice: 2
20 popped from stack.

Choose an operation:
```

```
9. Exit
Enter your choice: 2
20 popped from stack.

Choose an operation:
1. Stack: Push
2. Stack: Pop
3. Stack: Peek
4. Stack: Display
5. Queue: Enqueue
6. Queue: Dequeue
7. Queue: Peek
8. Queue: Display
9. Exit
Enter your choice: 3
Top element: 10

Choose an operation:
1. Stack: Push
2. Stack: Pop
3. Stack: Peek
4. Stack: Display
5. Queue: Enqueue
6. Queue: Dequeue
7. Queue: Peek
8. Queue: Display
9. Exit
Enter your choice: 4
Stack: 10

Choose an operation:
1. Stack: Push
2. Stack: Pop
3. Stack: Peek
4. Stack: Display
5. Queue: Enqueue
6. Queue: Dequeue
7. Queue: Peek
8. Queue: Display
9. Exit
Enter your choice: 8
Queue is empty.

Choose an operation:
```

LABORATORY PROGRAM – 9

WAP to Implement doubly link list with primitive operations

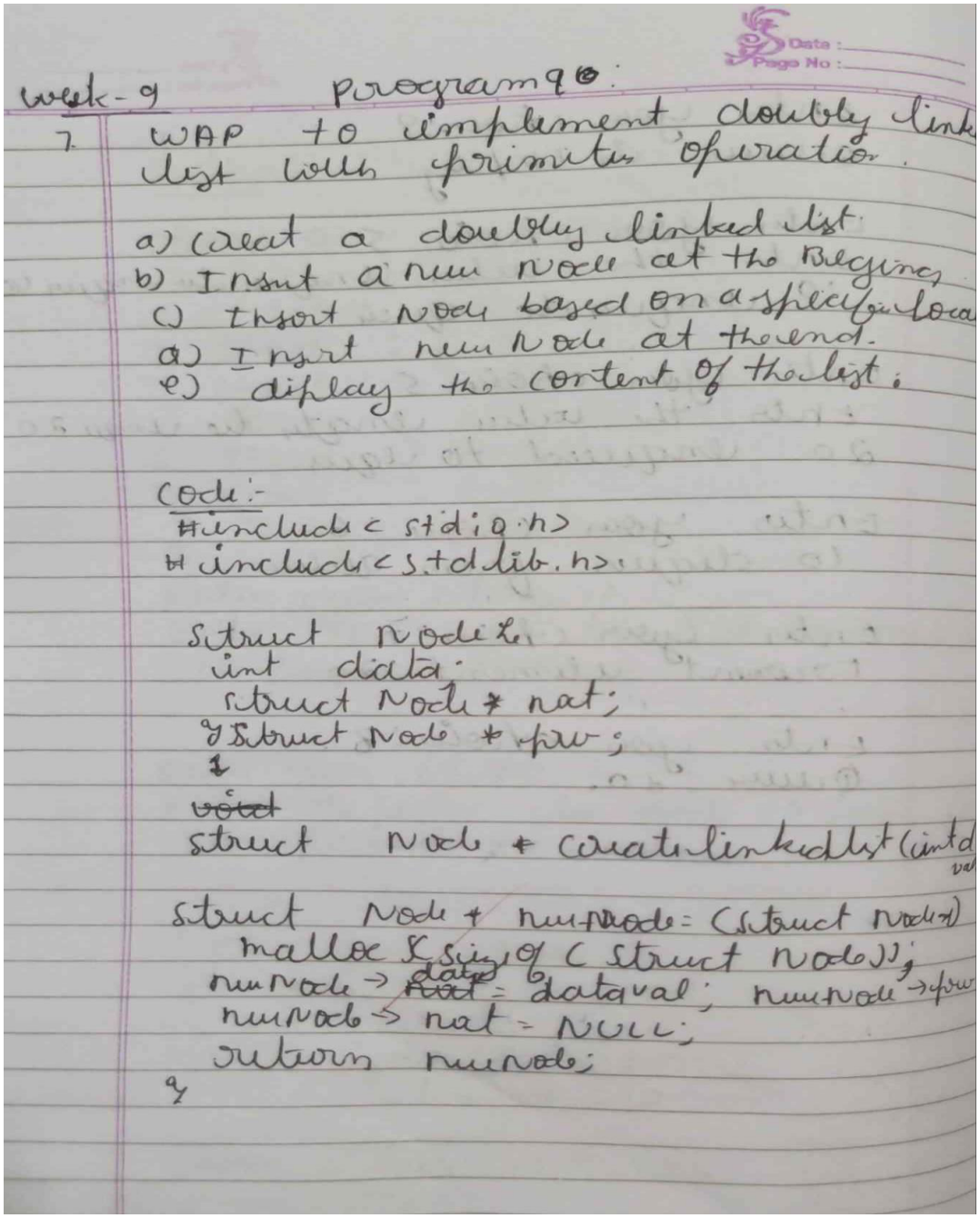
a) Create a doubly linked list.

b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

d) Display the contents of the list

OBSERVATION :



week-9 Program 9:

7. WAP to implement doubly link list with primitive operations.

a) Creat a doubly linked list
b) Insert a new node at the Begining
c) Insert node based on a specific loca
d) Insert new node at the end.
e) diplay the content of the list.

Code:-

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

struct Node{
 int data;
 struct Node * nat;
 struct Node * prv;
};
void
struct Node * createlinkedlist(int d
va

struct Node * newNode=(struct Node*)
   malloc(sizeof(struct Node));
newNode->data=dataval; newNode->prv
newNode->nat=NULL;
return newNode;
}
```

```
void insertAtBegin(struct node **
headref, int data){

struct node * newNode = createLinked
                        list(data);
 newNode->next = *headref;
 *headref->prev = newNode;
  *headref = newNode;
}

void insertAtEnd(struct node **
headref, int data){

struct Node * newNode = createLinked
                        list(data);

struct Node * temp = *headref;
while(temp->next->next != NULL){
   temp = temp->next;
  }
 temp->next = newNode;
 newNode->prev = temp;
 }
}

void insertAtposition(struct Node **
headref, int dataval, int position){

~~Sdbs~~ struct Node *newNode = createLink
list(dataval);
```

```
{ struct node * temp = * headref
for(int i=0; i<position; i++){
    temp = temp -> next;
}
temp new_node -> prev = temp;
    new_node -> next = temp -> ne
    temp -> next = new_node;
    temp -> next -> next = new_node;
}
}

void display list (struct node *
headref){

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

void main(){
struct node * head = NULL
int choice, val, position;
for while(1){
```

```
printf("Menu");
printf("1. Insut At Begin");
printf("2. Insert At end");
printf("3. Insert At position");
printf("4. Display");
printf("Enter your choice");
scanf("%d", &choice);

switch(choice)

case 1:
printf("Enter the value for Begin");
printf("
scanf("%d", &val);
insertAtBegin(&head, val);
break;

case 2:
printf("Enter the value for end");
scanf("%d" &val);
insutAtEnd(&head, val);
break;

case 3:
print("Enter the value for position
scanf("%d, &val);
printf("Enter the position");
scanf("%d", &position);
InsertAtposition(&head, data,
position);
break;
case 4:
display(&head);
```

```
case 5:
        printf("Exiting");
        exit(0);
case 6:
default:
        printf("Enter the perfect choice");
}
}
```

output:

1. Insert At the Beginning
2. Insert At the End
3. Insert A specific position
4. Display list.
5. exit

Enter your choice: 1;
Enter the data 10.

Enter your choice 2.
Enter the data 20

Enter your choice 3
Enter the data 30
Enter the position 3

Enter your choice 4
10 -> 20 -> 30.

CODE :

```
 #include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};
struct Node* createnode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
void insertatbegin(struct Node** head, int data) {
    struct Node* newNode = createnode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }}
void insertatend(struct Node** head, int data) {
    struct Node* newNode = createnode(data);
```

```
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }}
void insertatposition(struct Node** head, int data, int position) {
    struct Node* newNode = createnode(data);
    if (position == 1) {
        insertatbegin(head, data);
        return;
    }
    struct Node* temp = *head;
     for(int i=2;i<position;i++){
         temp = temp->next;
     }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
```

```
}
void display(struct Node* head) {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("List is empty.\n");
        return;   }
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("\n"); }
int main() {
    struct Node* head = NULL;
    int choice, data, position;
    while (1) {
        printf("1. Insert at the beginning\n");
        printf("2. Insert at the end\n");
        printf("3. Insert at a specific position\n");
        printf("4. Display list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the data  ");
                scanf("%d", &data);
```

```
            insertatbegin(&head, data);
            break;
        case 2:
            printf("Enter the data  ");
            scanf("%d", &data);
            insertatend(&head, data);
            break;
        case 3:
            printf("Enter the data: ");
            scanf("%d", &data);
            printf("Enter the position : ");
            scanf("%d", &position);
            insertatposition(&head, data, position);
            break;
        case 4:
            display(head);
            break;
        case 5:
            printf("Exit\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

OUTPUT :

```
PS D:\3rd sem\DSA\lab programs> ./a.exe
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display list
5. Exit
Enter your choice: 1
Enter the data  10
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display list
5. Exit
Enter your choice: 2
Enter the data  20
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display list
5. Exit
Enter your choice: 3
Enter the data: 30
Enter the position : 3
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display list
5. Exit
Enter your choice: 4
10 -> 20 -> 30 ->
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display list
5. Exit
Enter your choice:
```

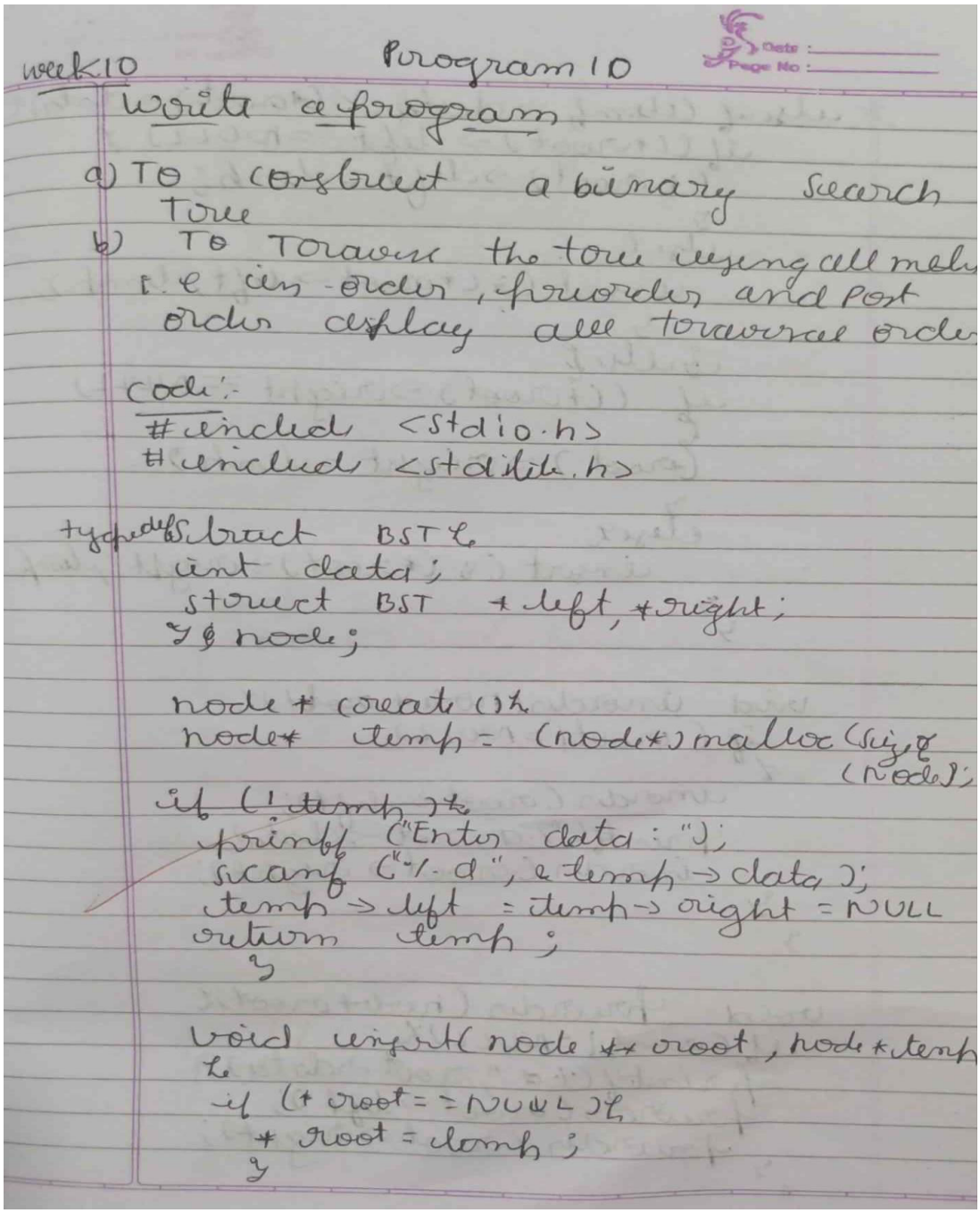
LABORATORY PROGRAM – 10

Write a program a) To construct a binary Search tree.

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

c) To display the elements in the tree

OBSERVATION :



week 10 Program 10

Write a program

a) To construct a binary search tree

b) To traverse the tree using all meth i.e in-order, preorder and post order display all traversal orde

code:

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

typedef struct BST {
    int data;
    struct BST *left, *right;
} node;

node * create () {
    node* temp = (node*) malloc (sizeof (Node));
if (!temp) {
    printf ("Enter data : ");
    scanf ("%d", &temp -> data );
    temp -> left = temp -> right = NULL
    return temp;
    }

void insert( node ** root , node *temp
{
    if (*root == NULL ){
        * root = temp ;
    }
```

```
else if (temp -> data < (*root) -> data)
    if ((*root) -> left == NULL) {
        (*root) -> left = temp;
    }
    else {
        insert (&(*root -> left, temp);
    }
else {
    if ((*root) -> right == NULL)
    {
        (*root) -> right = temp;
    }
    else {
        insert (&(*root) -> right, temp
    }
}

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

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

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

int main(){
  node *root = NULL;
  node *temp;
  char choice = 'y';

while(choice == 'y' || choice == 'Y'){
  temp = create();
  insert(&root, temp);
  printf("DO you want to enter more no
  (y/n)");
  scanf("%c", &choice);
}

printf("\n In-order Traversal");
inorder(root);

printf("\n pre-order Traversal");
preOrder(root);

printf("\n Post order Traversal");
postOrder(root);

return 0;
}
```

output:-

Enter the data : 10
DO you want Enter mor Node.
Y

Enter the data : 5
DO you want Enter mor Node:
Y

Enter the data : 7
DO you wants Enter mor Node:
Y

Enter the data : 13
DO you want Ent mor Node:
Y

Enter the data 15
DO you wans Ents mor Node.
Y/N

Inorder Traversal 5 7 10 13 15

pre - Oder Traversal 10 5 7 13 15

Postorder Traversed 7 5 15 13 10

CODE :

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

typedef struct BST
{
   int data;
   struct BST *left, *right;
} node;

node *createNode()
{
   node *temp = (node *)malloc(sizeof(node));
   printf("Enter the data : ");
   scanf("%d", &temp->data);
   temp->left = temp->right = NULL;
   return temp;
}

void insert(node **root, node *temp)
{
   if (*root == NULL)
   {
      *root = temp;
   }
   else if (temp->data < (*root)->data)
```

```
    {
        if ((*root)->left == NULL)
        {
            (*root)->left = temp;
        }
        else
        {
            insert(&(*root)->left, temp);
        }
    }
    else
    {
        if ((*root)->right == NULL)
        {
            (*root)->right = temp;
        }
        else
        {

            insert(&(*root)->right, temp);
        }
    }
}

void inorder(node *root)
{
```

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

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

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

```
}

int main()
{
   node *root = NULL;
   node *temp;
   char choice = 'y';

   while (choice == 'y' || choice == 'Y')
   {
      temp = createNode();
      insert(&root, temp);
      printf("Do you want to Enter More node : \n");
      scanf(" %c", &choice);
   }

   printf("\n Inorder Traversal");
   inorder(root);

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

   printf("\n Post-order Traversal");
   postOrder(root);
}
```

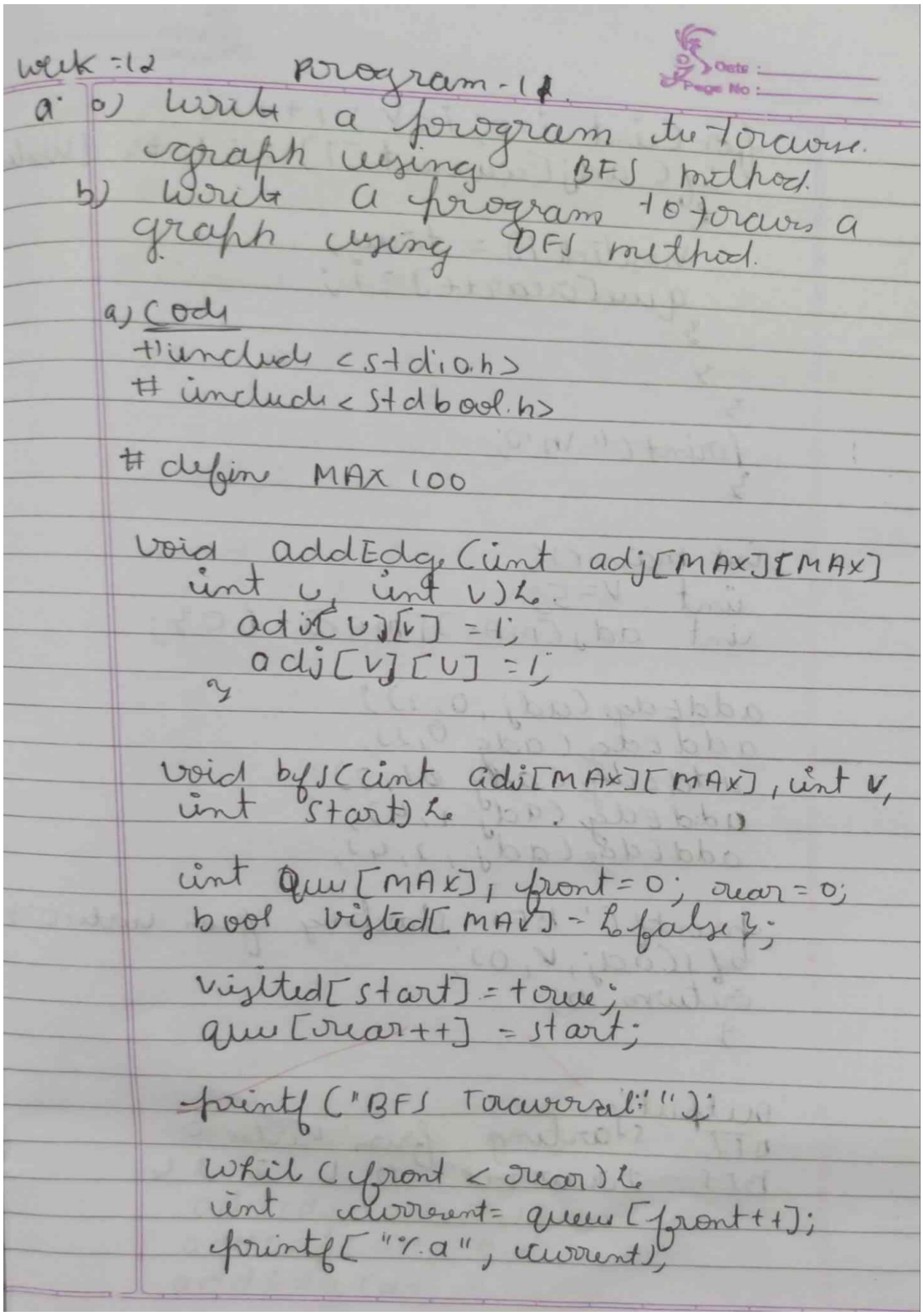
OUTPUT :

```
PS D:\3rd sem\DSA\lab programs> ./a.exe
Enter the data : 10
Do you want to Enter More node :
y
Enter the data : 5
Do you want to Enter More node :
y
Enter the data : 7
Do you want to Enter More node :
y
Enter the data : 13
Do you want to Enter More node :
y
Enter the data : 15
Do you want to Enter More node :
y
Enter the data : 10
Do you want to Enter More node :
n

 Inorder Traversal 5  7  10  10  13  15
 Pre-Order Traversal 10  5  7  13  10  15
 Post-order Traversal 7  5  10  15  13  10
PS D:\3rd sem\DSA\lab programs>
```

LABORATORY PROGRAM – 11

Write a program to traverse a graph using BFS method.

OBSERVATION :



Week :12 Program - 11

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

a) Code

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

#define MAX 100

void addEdge(int adj[MAX][MAX]
  int u, int v){
    adj[u][v] = 1;
    adj[v][u] = 1;
}

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

int que[MAX], front=0; rear = 0;
bool visited[MAX] = {false};

visited[start] = true;
que[rear++] = start;

printf("BFS Traversal:");

while(front < rear){
int current= que[front++];
printf("%d", current);
```

```
for (int i=0; i<V; i++){
  if( adj[current[i] == 1 && !visi
  {
    visited[i] = true;
    que[rear++] = i;
  }
  }
}
printf("\n");
}

int main(){
 int V=5;
 int adj[MAX][MAX]={0};

addEdge(adj,0,1);
addEdg(adj 0,2);
addEdg(adj 0,3);
addEdg(adj 1,4);
addEdge(adj,2,4);

printf(" BFS starting from vertex 0:
bfs(adj,V,0);
return 0;
}
```

Output:-
BFS starting from vertex 0:
BFS traversal 0 1 2 3 4

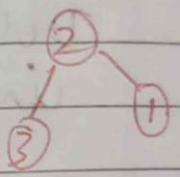
b) code:

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

#define MAX 100

void addEdge(int adj[MAX][MAX],
  int u, int v){
  adj[u][v]=1;
  adj[v][u]=1;

void dfs(int adj[MAX][MAX],
int v, bool visited[MAX]){

  visited[v]=true;
  printf("%d", v);

for(int i=0; i<MAX; i++){
  if(adj[v][i]==1 && !visited[i]){
    dfs(adj, i, visited);
  }
}

int main(){
  int v=5;
int adj[MAX][MAX]={0};
  bool visited[MAX]={false};

  addEdge(adj, 0, 1);
  addEdge(adj, 0, 2);
  addEdge(adj, 0, 3);
```

addEdge(adj, 1, 4);
addEdge(adj, 2, 4);

printf("DFS starting from Vertex 0
dfs(adj, 0, visited);
return 0;
}

Output:-
DFS starting from Vertex 0
0 1 4 2 3



CODE :

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100
void addEdge(int adj[MAX][MAX], int u, int v) {
    adj[u][v] = 1;
    adj[v][u] = 1;
}
void dfs(int adj[MAX][MAX], int v, bool visited[MAX]) {
    visited[v] = true;
    printf("%d ", v);

    for (int i = 0; i < MAX; i++) {
        if (adj[v][i] == 1 && !visited[i]) {
            dfs(adj, i, visited);
        }
    }
}

int main() {
    int V = 5;
    int adj[MAX][MAX] = {0};
    bool visited[MAX] = {false};

    addEdge(adj, 0, 1);
    addEdge(adj, 0, 2);
```

```
    addEdge(adj, 0, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 4);

    printf("DFS starting from vertex 0:\n");
    dfs(adj, 0, visited);

    return 0;
}
```

OUTPUT :