VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SAMIR CHAUDHARY (1BM23CS294)

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 **SAMIR CHAUDHARY (1BM23CS294),** who is bonafide student of **B.M.S.College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering inComputer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Dr. kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

		b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list. B)Program - Leetcode platform(739) Daily Temperature.	
6	**02/12**	A) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists. B) WAP to Implement Single Link List to simulate Stack & Queue Operations.	70-85
7	**16/12**	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	86-94
8	**23/12**	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.	95-101
9	**23/12**	A) Write a program to traverse a graph using BFS method. B)Write a program to check whether given graph is connected or not using DFS method.	102-112
10			

Github Link: https://github.com/SAMIR-1BM23CS294/Data-Structure

Program 1

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

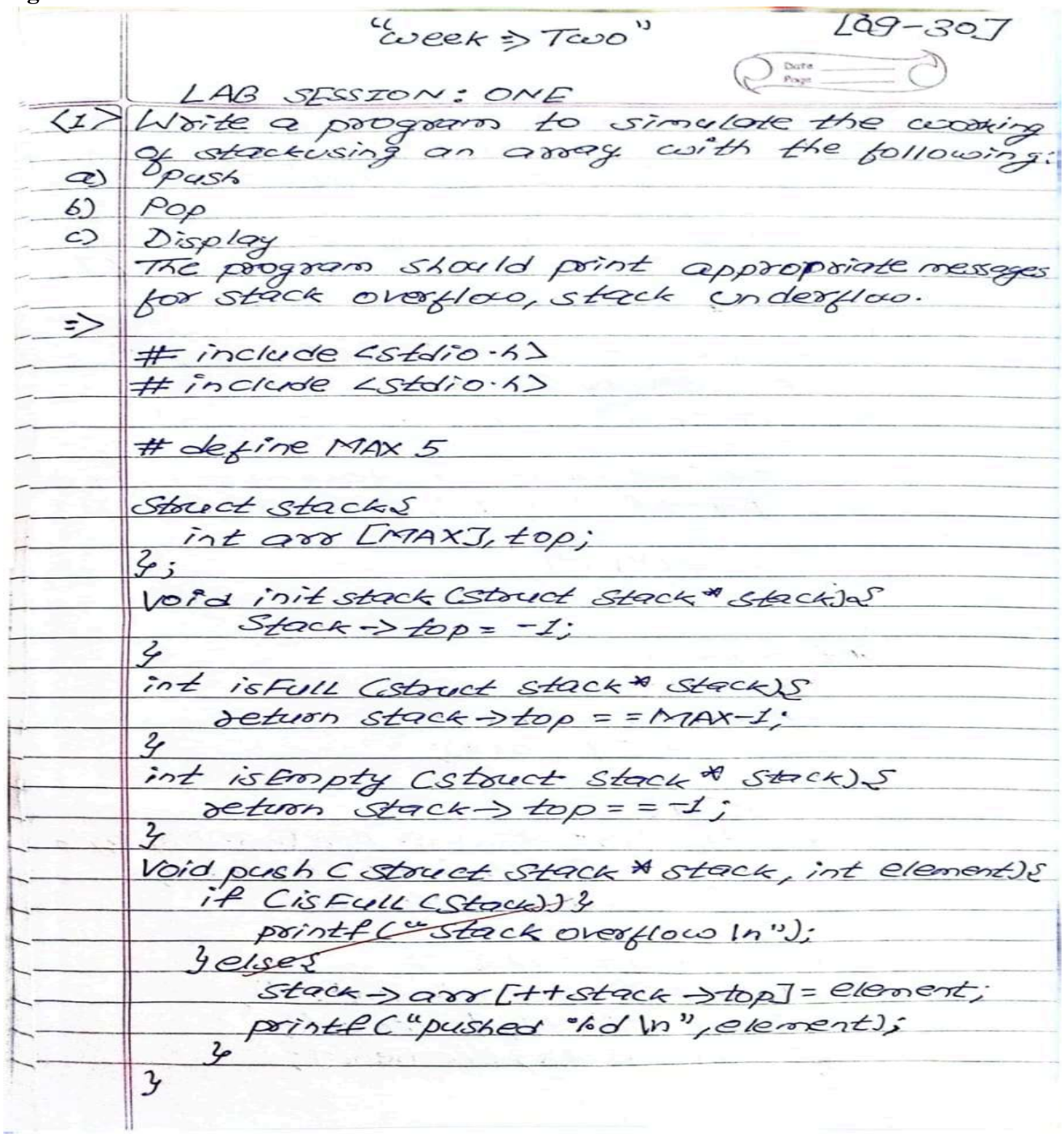
a) Push

b) Pop

c) Display

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

Algorithm:



"Week => Two" [09-30]

LAB SESSION: ONE

(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.

=>

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

#define MAX 5

Struct stack{
   int arr[MAX], top;
};
Void initstack(struct stack* stack){
      Stack->top = -1;
}
int isFull(struct stack* stack){
    return stack->top == MAX-1;
}
int isEmpty(struct stack* stack){
    return stack->top == -1;
}
Void push(struct stack* stack, int element){
    if(isFull(Stack)){
        printf("stack overflow\n");
    }else{
        stack->arr[++stack->top] = element;
        printf("pushed %d\n", element);
    }
}
```

```
void pop (struct stack* stack){
  if (isEmpty (stack)){
    printf ("Stack Underflow\n");
  }else{
    printf ("popped %d\n",
        stack->arr [stack->top--]);
  }
}
Void display (struct stack* stack){
  if (isEmpty (stack)){
    printf ("stack is empty\n");
  }else{
    for (int i=0; i<=stack->top; i++){
      printf ("%d", stack->arr [i]);
    }
    printf("\n");
  }
}
int main (){
  struct stack stack;
  int choice, element;
  initstack (&stack);

  while (1){
    printf ("\n1. Push\n2. pop\n3. Display\n
          4. Exit\n Enter choice: ");
    scanf ("%d", &choice);

    if (choice == 1){
      printf ("Enter element: ");
      scanf ("%d", &element);
      push (&stack, element);
```

```
} elseif (choice == 2) {
    pop (&stack);
} elseif (choice == 3) {
    display (&stack);
} elseif (choice == 4) {
    printf("Exiting ---\n");
    break;
} else {
    printf("Invalid choice\n");
}
return 0;
}
```

Output:-

```
1. Push
2. pop
3. Display
4. Exit
Enter choice : 1
Enter element : 10
Pushed 10

1. push
2. pop
3. Display
4. Exit
Enter choice : 1
Enter element: 20
Pushed 20
```

1. Pushed
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 30
pushed 30

1. Pushed
2. Pop
3. Display
4. Exit
Enter choice: 3
10 20 30

1. Pushed
2. Pop
3. Display
4. Exit
Enter choice: 2
popped 30

1. Pushed
2. Pop
3. Display
4. Exit
Enter choice: 3
10 20

Enter choice: 4
Exiting---

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
struct Stack {
   int arr[MAX], top;
};
void initStack(struct Stack *stack) {
   stack->top = -1;
}

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

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

void push(struct Stack *stack, int element) {
   if (isFull(stack)) {
      printf("Stack Overflow\n");
   } else {
      stack->arr[++stack->top] = element;
      printf("Pushed %d\n", element);
   }
}

void pop(struct Stack *stack) {
   if (isEmpty(stack)) {
      printf("Stack Underflow\n");
   } else {
      printf("Popped %d\n", stack->arr[stack->top--]);
   }
}

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

```
int main() {
    struct Stack stack;
    int choice, element;
    initStack(&stack);

    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter element: ");
            scanf("%d", & element);
            push(&stack, element);
        } else if (choice == 2) {
            pop(&stack);
        } else if (choice == 3) {
            display(&stack);
        } else if (choice == 4) {
            printf("Exiting...\n");
            break;
        } else {
            printf("Invalid choice\n");
        }
    }

    return 0;
}
```

Output:

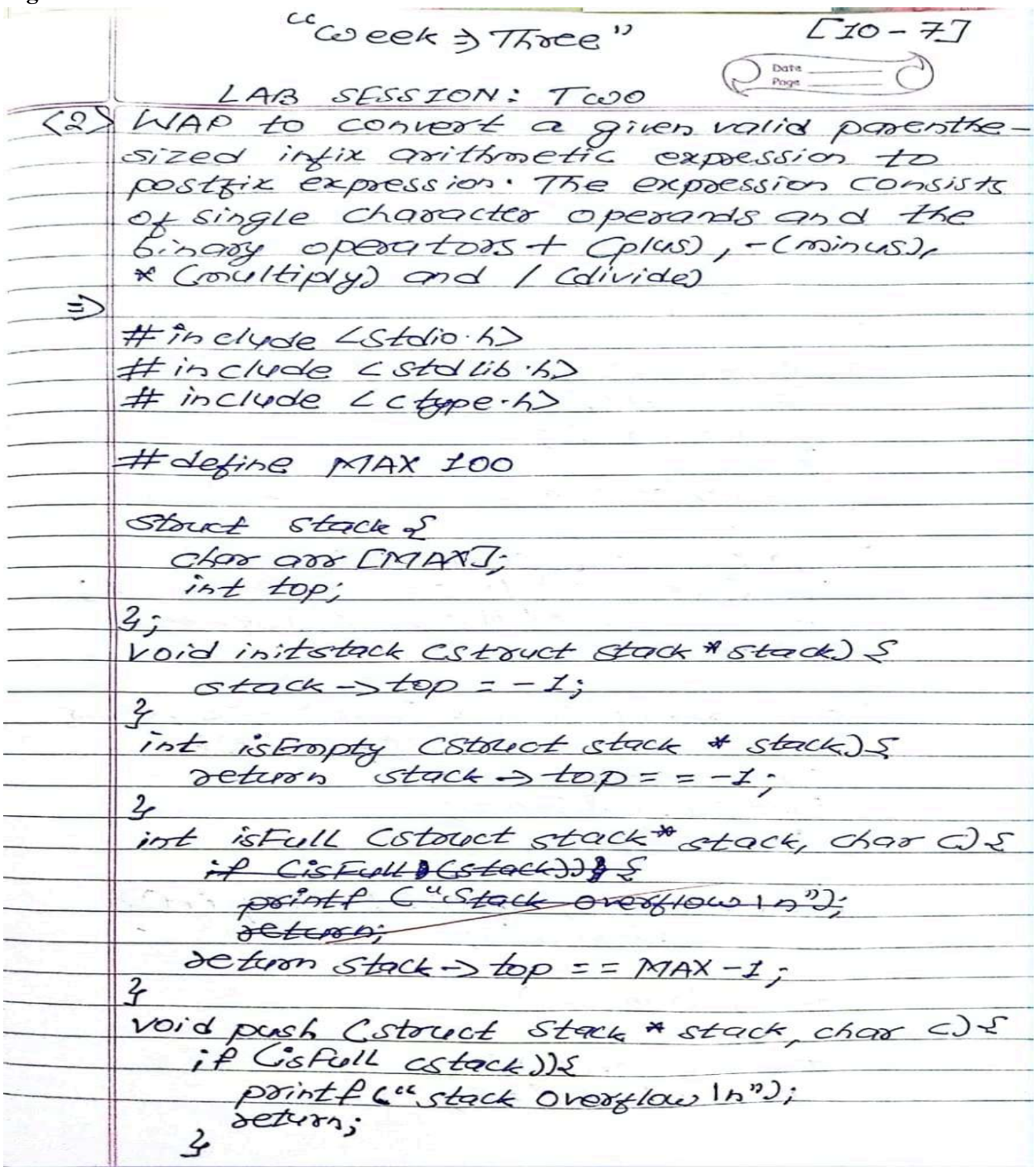
```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 10
Pushed 10

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 20
Pushed 20

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 30
Pushed 30

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
10 20 30
```

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Popped 30

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
10 20

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

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).

Algorithm:



```
"Week 3 Three"                                   [10-7]
LAB SESSION: Two
(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)
=>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define MAX 100

struct stack {
    char arr[MAX];
    int top;
};
void initstack (struct stack *stack) {
    stack->top = -1;
}
int isEmpty (struct stack * stack) {
    return stack->top == -1;
}
int isFull (struct stack* stack, char c) {
    return stack->top == MAX-1;
}
void push (struct stack * stack, char c) {
    if (isFull (stack)) {
        printf ("stack overflow \n");
        return;
    }
```

```
    stack->arr[++stack->top] = c;
}
char pop(struct stack *stack){
    if (isEmpty(stack)) {
        return -1;
    }
    return stack->arr[stack->top--];
}
char peek(struct stack *stack) {
    if (isEmpty(stack)) {
        return -1;
    }
    return stack->arr[stack->top];
}
int precedence(char c){
    if (c == '+' || c == '-') return 1;
    if (c == '*' || c == '/') return 2;
    return 0;
}
Void infixTOpostfix(char *infix, char
                        *postfix){
    struct stack stack;
    initstack(&stack);
    int k=0;

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

if(isalnum(c)){

if(isalnum(c)){
    postfix[k++] = c;
```

```
    } else if (c == '(') {
        push(&stack, c);
    } else if (c == ')') {
        while (!isEmpty(&stack) && peek(&stack) !=
                                        '(') {
            postfix[k++] = pop(&stack);
        }
        pop(&stack);
    } else if (c == '+' || c == '-' || c == '*' ||
                                    c == '/') {
        while (!isEmpty(&stack) && precedence
            (peek(&stack)) >= precedence(c)) {
            postfix[k++] = pop(&stack);
        }
        push(&stack, c);
    }
}
while (!isEmpty(&stack)) {
    postfix[k++] = pop(&stack);
}
postfix[k] = '\0';
}
int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter a valid infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);
    printf("postfix expression: %s\n", postfix);
    return 0;
}
```

Output:-

Case: 1 Enter a valid infix expression: (a+b)*(c-d)

Postfix expression: ab+cd-*

case:2 Enter a valid infix expression: a*(b+c)/d

Postfix expression: abc+*d/

case:3 Enter a valid infix expression: a+b*c

Postfix expression: abc*+

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX 100
struct Stack {
   char arr[MAX];
   int top;
};
void initStack(struct Stack *stack) {
   stack->top = -1;
}

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

int isFull(struct Stack *stack) {
   return stack->top == MAX - 1;
}
void push(struct Stack *stack, char c) {
   if (isFull(stack)) {
      printf("Stack overflow\n");
      return;
   }
   stack->arr[++stack->top] = c;
}

char pop(struct Stack *stack) {
   if (isEmpty(stack)) {
      return -1;
   }
   return stack->arr[stack->top--];
}

char peek(struct Stack *stack) {
   if (isEmpty(stack)) {
      return -1;
   }
   return stack->arr[stack->top];
}

int precedence(char c) {
   if (c == '+' || c == '-') return 1;
   if (c == '*' || c == '/') return 2;
   return 0;
}
void infixToPostfix(char *infix, char *postfix) {
```

```
    struct Stack stack;
    initStack(&stack);
    int k = 0;

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

        if (isalnum(c)) {
            postfix[k++] = c;
        } else if (c == '(') {
            push(&stack, c);
        } else if (c == ')') {
            while (!isEmpty(&stack) && peek(&stack) != '(') {
                postfix[k++] = pop(&stack);
            }
            pop(&stack);
        } else if (c == '+' || c == '-' || c == '*' || c == '/') {
            while (!isEmpty(&stack) && precedence(peek(&stack)) >= precedence(c)) {
                postfix[k++] = pop(&stack);
            }
            push(&stack, c);
        }
    }
    while (!isEmpty(&stack)) {
        postfix[k++] = pop(&stack);
    }

    postfix[k] = '\0';
}
int main() {
    char infix[MAX], postfix[MAX];

    printf("Enter a valid infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

Output:

```
Enter a valid infix expression: (a+b)*(c-d)
Postfix expression: ab+cd-*
```

Program 3

A) 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.

Algorithm:

"Week 3 Four" [10-14]

LAB SESSION : THREE (Part-A)

(3) Write a program to stimulate 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.

=>

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

#define MAX 5 //Maximum size of the queue

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

// Function to create a queue
Queue* CreateQueue(){
   Queue* q = (Queue*)malloc(sizeof(Queue));
    q->front = -1;
    q-> rear = -1;
    return q;
}

// Function to check if the queue is full
int isFull(Queue* q){
   return q-> rear == MAX-1;
}
```

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

// Function to insert an element into the queue
void enqueue(Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue overflow! Unable to
                insert %d\n", value);
        return;
    }
    if (q->front == -1) {
        q->front = 0; // Initialize front for the
                      first element
    }
    q->rear++;
    q->items[q->rear] = value;
    printf("%d inserted into the queue.\n", value);
}

// Function to delete an element from the queue
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue empty! Unable to delete an
                element.\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = -1; // Reset the queue
```

```
        q->rear = -1;
    }
    printf("%d deleted from the queue.\n",
                                    item);
    return item;
}

// Function to display the elements of queue
void display(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d", q->items[i]);
    }
    printf("\n");
}

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

    do {
        printf("\n Queue Menu:\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 an integer to insert: ");
    scanf("%d", &value);
    enqueue(q, value);
    break;
case 2:
    dequeue(q);
    break;
case 3:
    display(q);
    break;
case 4:
    printf("Exiting---\n");
    break;
default:
    printf("Invalid choice! Please try again.\n");
}
} while (choice != 4);

free(q); // free the allocated memory for the queue

return 0;
}
```

Seen
04/10/24

Output:-

Queue menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
~~Enter the value to insert: 10~~
Enter an integer to insert: 10
10 inserted into the queue.

Queue menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 20
20 inserted into the queue.

Queue menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20

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

Queue menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements : 20

Queue menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting the program.

Code:

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

#define MAX 5

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

Queue* CreateQueue() {
   Queue* q = (Queue*)malloc(sizeof(Queue));
   q->front = -1;
   q->rear = -1;
   return q;
}

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

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

void enqueue(Queue* q, int value) {
   if (isFull(q)) {
      printf("Queue Overflow! Unable to insert %d.\n", value);
      return;
   }
   if (q->front == -1) {
      q->front = 0;
   }
   q->rear++;
   q->items[q->rear] = value;
   printf("%d inserted into the queue.\n", value);
}

int dequeue(Queue* q) {
   if (isEmpty(q)) {
      printf("Queue Empty! Unable to delete an element.\n");
      return -1;
   }
   int item = q->items[q->front];
   q->front++;
```

```
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    printf("%d deleted from the queue.\n", item);
    return item;
}

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

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

    do {
        printf("\nQueue Menu:\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 an integer to insert: ");
                scanf("%d", &value);
                enqueue(q, value);
                break;
            case 2:
                dequeue(q);
                break;
            case 3:
                display(q);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
```

```
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

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

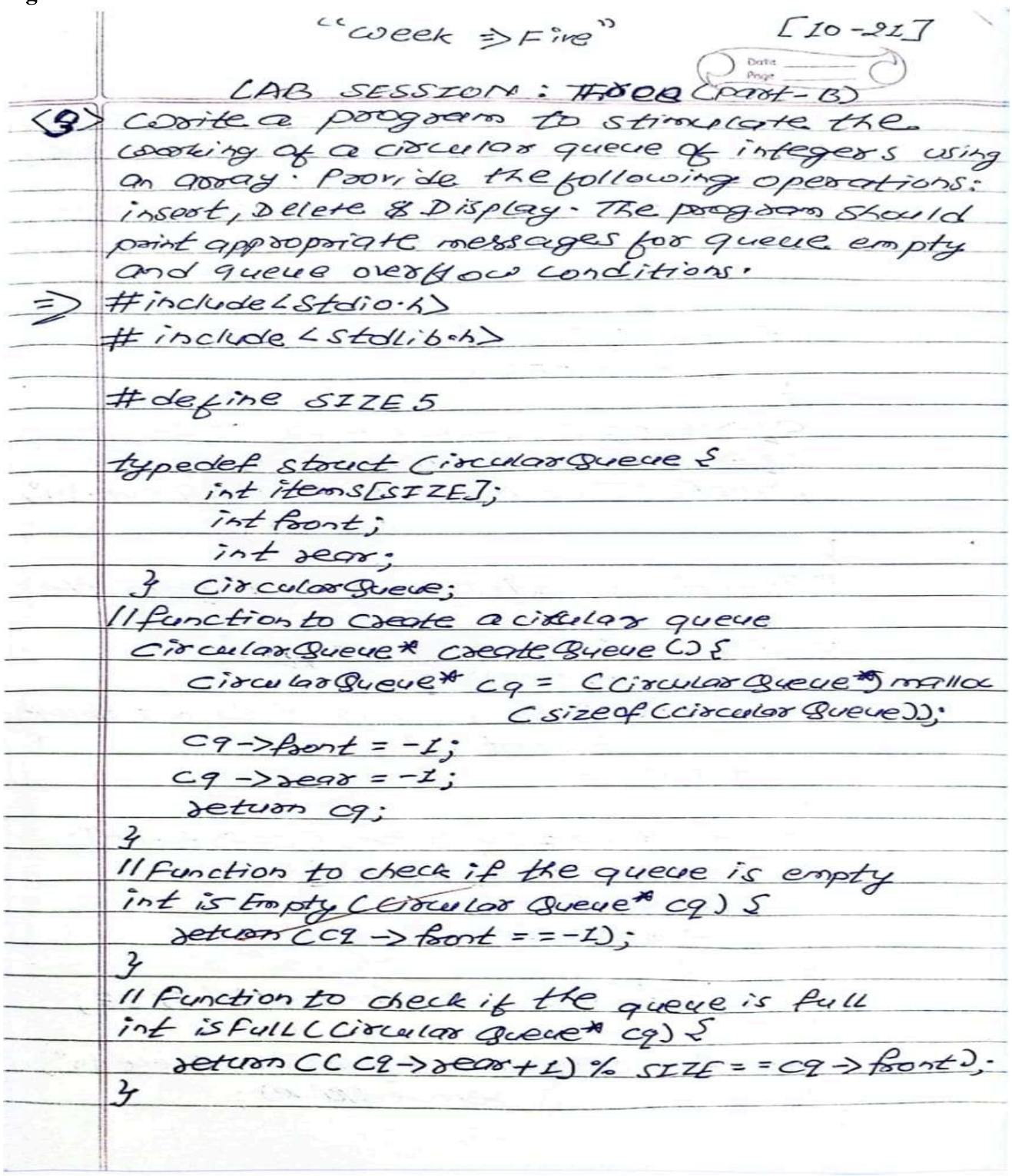
Output:

```
Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 10
10 inserted into the queue.

Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 20
20 inserted into the queue.

Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20

Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
10 deleted from the queue.
```

```
Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20

Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

B) 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.

Algorithm:



"Week ⇒ Five" [10-21]

LAB SESSION : 10 (Part-B)

(3) Write a program 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.

⇒

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

#define SIZE 5

typedef struct circularQueue {
    int items[SIZE];
    int front;
    int rear;
} circularQueue;
// function to create a circular queue
circularQueue* createQueue() {
    circularQueue* cq = (circularQueue*) malloc
                        (sizeof(circularQueue));
    cq->front = -1;
    cq->rear = -1;
    return cq;
}
// Function to check if the queue is empty
int isEmpty(circularQueue* cq) {
    return (cq->front == -1);
}
// Function to check if the queue is full
int isFull(circularQueue* cq) {
    return ((cq->rear + 1) % SIZE == cq->front);
}
```

```
// Function to insert an element into the queue
void insert(CircularQueue* cq, int value){
  if(isFull(cq)){
    printf("Queue Overflow: Cannot insert
            element.\n");
    return;
  }
  if(isEmpty(cq)){
    cq->front = 0;
  }
  cq->rear = (cq->rear+1) % SIZE;
  cq->items[cq->rear] = value;
  printf("Inserted %d into the queue.\n",
                value);
}
// Function to delete an element from the
queue
void delete(CircularQueue* cq){
  if(isEmpty(cq)){
    printf("Queue underflow: Cannot delete
            element.\n");
    return;
  }
int removedValue = cq->items[cq->front];
if(cq->front == cq->rear){
    cq->front = -1;
    cq->rear = -1;
} else{
    cq->front = (cq->front+1) % SIZE;
}
printf("Deleted %d from the queue.\n",
            removedValue);
}
```

```
// Function to display the queue elements
void display(CircularQueue *cq){
    if(isEmpty(cq)){
      printf("Queue is empty.\n");
      return;
    }
  printf("Queue elements are: ");
  int i=cq->front;
  while(1){
      printf("%d", cq->items[i]);
      if(i==cq->rear){
        break;
      }
      i=(i+1)%SIZE;
    }
  printf("\n");
}
// Main function to display the circular
queue operations
int main(){
      CircularQueue cq;
      initializeQueue(&cq);
      int choice, value;

  do{
          printf("\n Circular Queue menu: \n");
          printf("\n1. 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 integer to insert:");
    scanf("%d", &value);
    insert(&cq, value);
    break;
case 2:
    delete(&cq);
    break;
case 3:
    display(&cq);
    break;
case 4:
    printf("Exiting the program.\n");
    exit(0);
default:
    printf("Invalid choice.
            Please try again.\n");
    }
}
return 0 while(choice!=4);
free(cq);
return 0;
}
```

Output :-

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 10
Inserted 10 into the queue.

~~Circular Qu~~
Enter your choice: 1
Enter an integer to insert: 20
Inserted 20 into the queue.

Enter your choice: 3
Queue elements are: 10 20

Enter your choice: 2
Deleted 10 from the queue.

Enter your choice: 3
Queue elements are: 20

Enter your choice: 1
Enter an integer to insert: 30
Inserted 30 into the queue.

Enter your choice: 1
Enter an integer to insert: 40
Inserted 40 into the queue.

Enter your choice: 1
Enter an integer to insert: 50
Inserted 50 into the queue.

Enter your choice: 1
Enter an integer to insert: 60
Inserted 60 into the queue.

Enter your choice: 3
Queue elements are: 20 30 40 50

Enter your choice: 2
Deleted 20 from the queue.

Enter your choice: 3
Queue elements are: 30 40 50

Enter your choice: 2
Deleted 30 from the queue.

Enter your choice: 2
Deleted 40 from the queue.

Enter your choice: 2
Deleted 50 from the queue

Enter your choice: 2
Queue underflow: Cannot delete element.

Enter your choice: 4
Exiting the program.

Code:

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

#define SIZE 5

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

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

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

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

void insert(CircularQueue* cq, int value) {
   if (isFull(cq)) {
      printf("Queue Overflow: Cannot insert element.\n");
      return;
   }
   if (isEmpty(cq)) {
      cq->front = 0;
   }
   cq->rear = (cq->rear + 1) % SIZE;
   cq->items[cq->rear] = value;
   printf("Inserted %d into the queue.\n", value);
}

void delete(CircularQueue* cq) {
   if (isEmpty(cq)) {
      printf("Queue Underflow: Cannot delete element.\n");
      return;
   }
   int removedValue = cq->items[cq->front];
   if (cq->front == cq->rear) {
```

```
        cq->front = -1;
        cq->rear = -1;
    } else {
        cq->front = (cq->front + 1) % SIZE;
    }
    printf("Deleted %d from the queue.\n", removedValue);
}

void display(CircularQueue* cq) {
    if (isEmpty(cq)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements are: ");
    int i = cq->front;
    while (1) {
        printf("%d ", cq->items[i]);
        if (i == cq->rear) {
            break;
        }
        i = (i + 1) % SIZE;
    }
    printf("\n");
}

int main() {
    CircularQueue* cq = createQueue();
    int choice, value;

    do {
        printf("\nCircular Queue Menu:\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 an integer to insert: ");
                scanf("%d", &value);
                insert(cq, value);
                break;
            case 2:
                delete(cq);
                break;
```

```
            case 3:
                display(cq);
                break;
            case 4:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

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

Output:

```
Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 10
Inserted 10 into the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 20
Inserted 20 into the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 10 20

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 10 from the queue.
```

```
Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 20

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 30
Inserted 30 into the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 40
Inserted 40 into the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 50
Inserted 50 into the queue.
```

```
Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter an integer to insert: 60
Inserted 60 into the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 20 30 40 50 60

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 20 from the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 30 40 50 60

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 30 from the queue.
```

```
Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 40 from the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 50 from the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 60 from the queue.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue Underflow: Cannot delete element.

Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

Program 4:

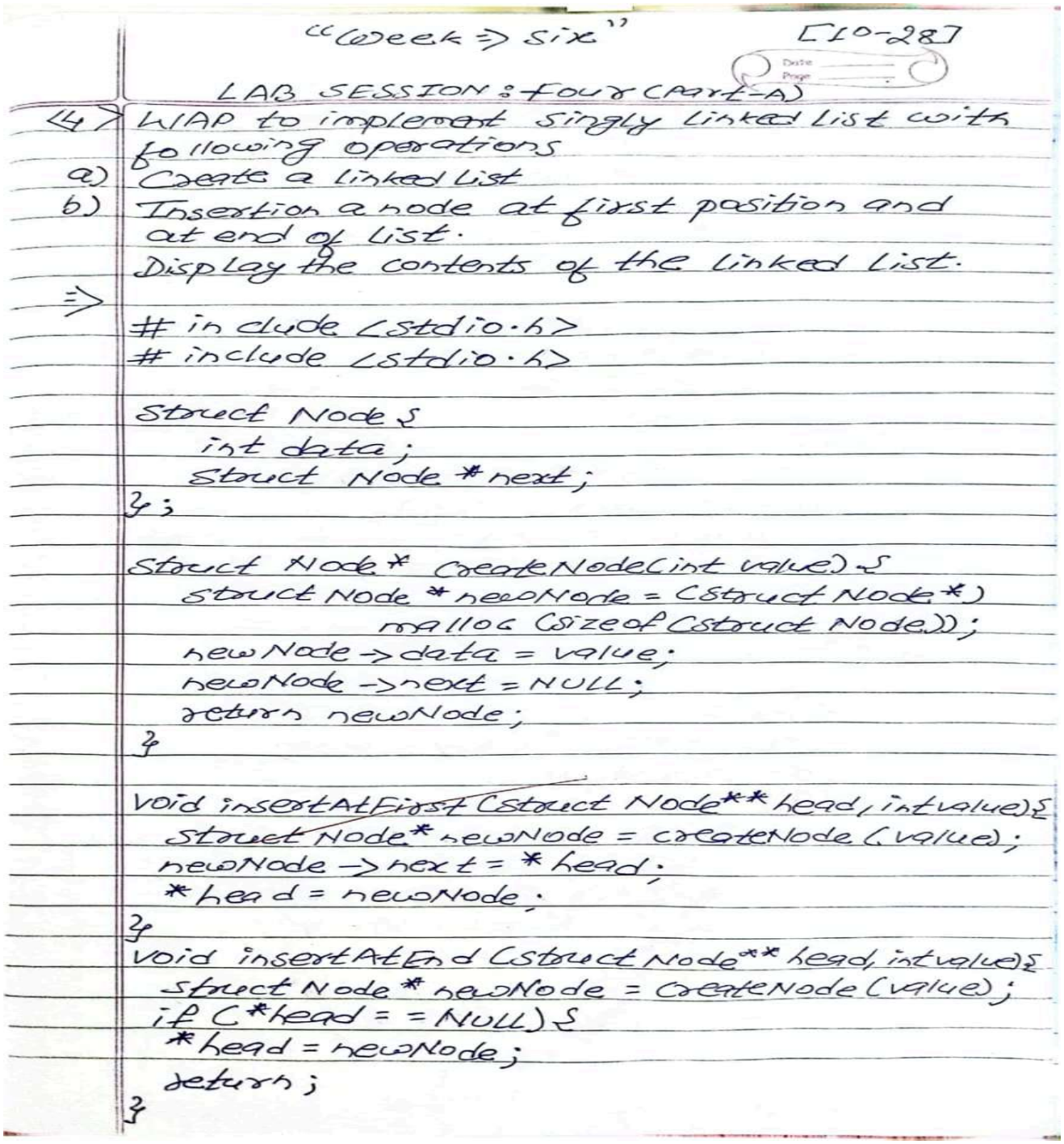
A)WAP to Implement Singly Linked List with following operations:

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

Algorithm:



"Week ⇒ Six" [10-28]

LAB SESSION : four (Part-A)

<4> WAP to implement singly linked list with following operations

a) Create a linked list

b) Insertion a node at first position and at end of list.

Display the contents of the linked list.

⇒

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

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

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

void insertAtFirst(struct Node** head, int value){
  struct Node* newNode = createNode(value);
  newNode->next = *head;
  *head = newNode;
}
void insertAtEnd(struct Node** head, int value){
  struct Node *newNode = createNode(value);
  if(*head == NULL){
   *head = newNode;
   return;
}
```

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

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

int main() {
    struct Node* head = NULL;
    int choice, value;
    do {
        printf("\n menu:\n");
        printf("1. Insert at first\n");
        printf("2. Insert at end\n");
        printf("3. Display list\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch(choice){
    case 1:
        printf("Enter value to insert
                at the beginning: ");
        scanf("%d", &value);
        insertAtFirst(&head, value);
        break;
    case 2:
        printf("Enter value to insert at the
                end: ");
        scanf("%d", &value);
        insertAtEnd(&head, value);
        break;
    case 3:
        printf("Contents of the linked list:
        displayList(head);
        break;
    case 4:
        printf("Exiting program...\n");
        break;
    default:
        printf("Invalid choice! Please try
                again.\n");
    }
} while(choice!=4);

  return 0;
}
```

Output:-

```
menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 1
Enter value to insert at the beginning : 10

menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 2
Enter value to insert at the end: 20

menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 3
Contents of the linked list: 10 -> 20 -> NULL

Enter your choice: 1
Enter value to insert at beginning : 5

Enter your choice: 2
Enter value to insert at end : 25

Enter your choice: 3
contents of the linked list: 5 -> 10 -> 20 -> 25 -> NULL
```

Code:

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


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


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


void insertAtFirst(struct Node** head, int value) {
   struct Node* newNode = createNode(value);
   newNode->next = *head;
   *head = newNode;
}


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

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

   temp->next = newNode;
}


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

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

    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Insert at first\n");
        printf("2. Insert at end\n");
        printf("3. Display list\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to insert at the beginning: ");
                scanf("%d", &value);
                insertAtFirst(&head, value);
                break;
            case 2:
                printf("Enter value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;
            case 3:
                printf("Contents of the linked list: ");
                displayList(head);
                break;
            case 4:
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

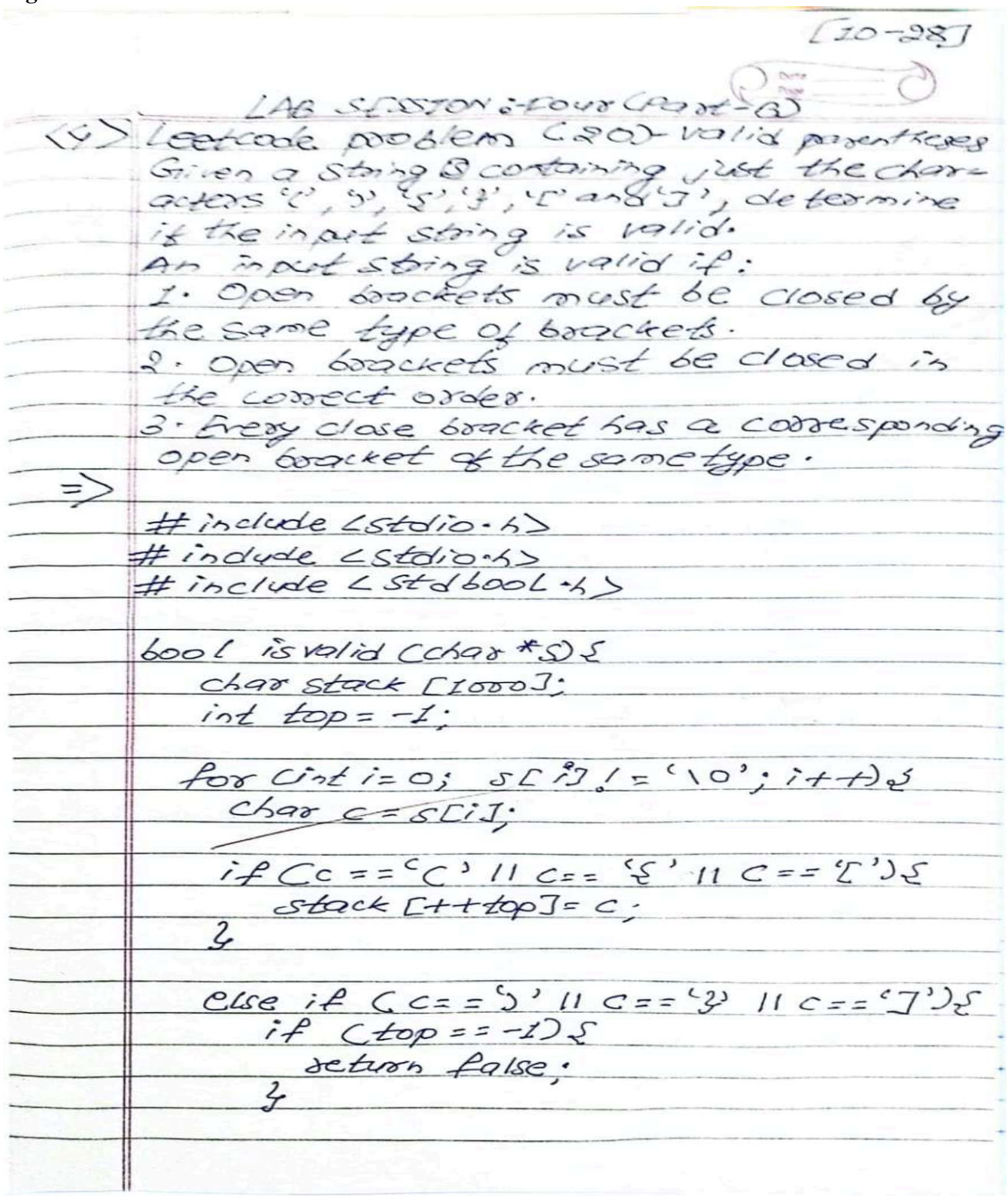
Output:

```
Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 10

Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 2
Enter value to insert at the end: 20

Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 3
Contents of the linked list: 10 -> 20 -> NULL
```

```
Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 1
Enter value to insert at the beginning: 5

Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 2
Enter value to insert at the end: 25

Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 3
Contents of the linked list: 5 -> 10 -> 20 -> 25 -> NULL

Menu:
1. Insert at first
2. Insert at end
3. Display list
4. Exit
Enter your choice: 4
Exiting program...
```

B)Program - Leetcode platform (Valid Parenthesis).

Algorithm:



[10-28]

LAB SESSION-FOUR (Part-B)

(i) Leetcode problem (20) Valid parentheses
Given a string S containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:
1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

=>

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

bool isValid(char *s){
    char stack[1000];
    int top = -1;

    for(int i=0; s[i]!='\0'; i++){
        char c=s[i];

        if(c=='(' || c=='{' || c=='['){
            stack[++top]=c;
        }

        else if(c==')' || c=='}' || c==']'){
            if(top==-1){
                return false;
            }
```

```
        char lastopened=stack[top--];

        if ((c==')' && lastopened!='(') ||
            (c=='}' && lastopened!='{') ||
            (c==']' && lastopened!='[')) {
            return false;
        }
    }
}
return top==-1;
}

int main(){
    char s[1000];

    printf("Enter a string of parentheses:");
    scanf("%s", s);

    if (isValid(s)) {
        printf("The string is valid.\n");
    } else {
        printf("The string is invalid.\n");
    }
    return 0;
}
```

Output :-

Enter a string of parentheses:
("Welcome to BMSCE").
The string is invalid.

Enter a string of parentheses:
("Samir")
The string is valid.

Code:

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

bool isValid(char *s) {
    char stack[1000];
    int top = -1;
    for (int i = 0; s[i] != '\0'; i++) {
        char c = s[i];
        if (c == '(' || c == '{' || c == '[') {
            stack[++top] = c;
        } else if (c == ')' || c == '}' || c == ']') {
            if (top == -1) {
                return false;
            }
            char lastOpened = stack[top--];

            if ((c == ')' && lastOpened != '(') ||
                (c == '}' && lastOpened != '{') ||
                (c == ']' && lastOpened != '[')) {
                return false;
            }
        }
    }

    return top == -1;
}
int main() {
    char s[1000];
   printf("Enter a string of parentheses: ");
    scanf("%s", s);

    if (isValid(s)) {
        printf("The string is valid.\n");
    } else {
        printf("The string is invalid.\n");
    }

    return 0;
}
```

Output:

```
Enter a string of parentheses: ("Welcome to BMSCE")
The string is invalid.
```

Program 5:

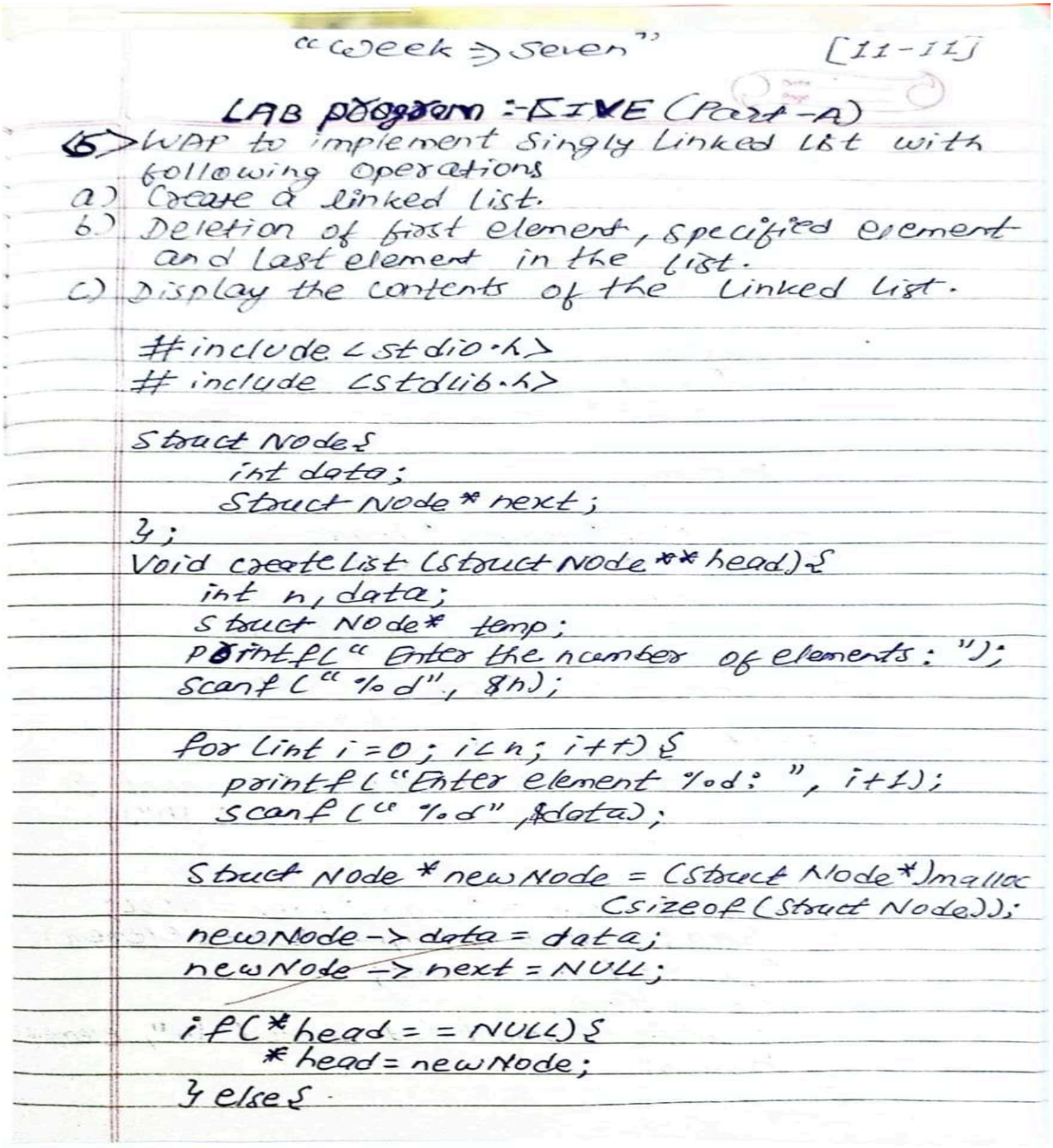
A)WAP to Implement Singly Linked List with following operations:

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.

Algorithm:



"Week ⇒ Seven" [11-11]

LAB program :- FIVE (Part-A)

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.

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

Struct Node{
    int data;
    Struct Node * next;
};
Void createList (Struct Node ** head){
    int n, data;
    Struct Node* temp;
    printf(" Enter the number of elements: ");
    scanf("%d", &n);

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

    Struct Node * newNode = (Struct Node*)malloc
                            (sizeof(Struct Node));
    newNode -> data = data;
    newNode -> next = NULL;

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

```
        temp = *head;
        while (temp->next != NULL){
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void deleteFirst (struct Node **head) {
    if(*head == NULL){
        printf("List is empty. No element to
                                delete.\n");
        return;
    }
    struct Node *temp = *head;
    *head = (*head)->next;
    free (temp);
    printf("First element deleted.\n");
}
void deleteSpecified(struct Node **head, int
                                element){
    if(*head == NULL){
        printf("List is empty. No element to
                                delete.\n");
        return;
    }
    struct Node *temp = *head, *prev = NULL;
    if (temp != NULL && temp->data == element){
        *head = temp->next;
        free (temp);
        printf("Element %d deleted.\n", element
        return;
}
```

```
while (temp != NULL && temp -> data != element){
    prev = temp;
    temp = temp -> Next;
}
if (temp == NULL){
    printf("Element %d not found. \n", element);
    return;
}
prev -> next = temp -> next;
free (temp);
printf("Element %d deleted. \n", element);
}
void deletelast (struct Node ** head){
  if (* head == NULL){
      printf("List is empty. No element to
                          delete. \n");
      return;
  }
  Struct Node * temp = * head, * prev = NULL;
  if (temp -> next == NULL){
      free (temp);
      * head = NULL;
      printf("Last element deleted - \n");
      return;
  }
  while (temp -> next != NULL){
      prev = temp;
      temp = temp -> next;
  }
  prev -> next = NULL;
  free (temp);
  printf(" last element deleted. \n");
}
```

```
Void display(struct Node * head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    Struct Node * temp = head;
    printf("Linked list contents: ");
    while (temp != NULL) {
        printf("%d -> ", temp -> data);
        temp = temp -> next;
    }
    printf("NULL\n");
}
int main() {
    Struct Node * head = NULL;
    int choice, element;
    while (1) {
        printf("\n Menu:\n");
        printf("1. Create Linked list\n");
        printf("2. Delete First Element\n");
        printf("3. Delete Specified Element\n");
        printf("4. Delete Last Element\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", & choice);

        Switch (choice) {
            case 1:
                createList(&head);
                break;
```

```
        case 2:
            deleteFirst(&head);
            break;
        case 3:
            printf("Enter the element to delete:
                                        ");
            scanf("%d", &element);
            deleteSpecified(&head, element);
            break;
        case 4:
            deleteLast(&head);
            break;
        case 5:
            display(head);
            break;
        case 6:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try
                                    again.\n");
    }
}
return 0;
}
```

Output :-

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice : 1
Enter the number of elements : 5
Enter element 1 : 10
Enter element 2 : 20
Enter element 3 : 30
Enter element 4 : 40
Enter element 5 : 50

Enter your choice : 2
First element deleted

Enter your choice : 5
Linked list elements : 20 -> 30 -> 40 -> 50
                       -> NULL

Enter your choice : 3
Enter the element to delete : 30
Element 30 deleted

Enter your choice : 5
Linked list contents : 20 -> 40 -> 50 -> NULL

Enter your choice : 4
Last element deleted.
```

Enter your choice: 5
Linked List contents: 20 → 40 → NULL

Enter your choice: 6
Exiting the program.

Code:

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

// Definition of a node in the linked list
struct Node {
   int data;
   struct Node* next;
};

// Function to create a linked list
void createList(struct Node** head) {
   int n, data;
   struct Node* temp;

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

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

      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
      newNode->data = data;
      newNode->next = NULL;

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

// Function to delete the first element
void deleteFirst(struct Node** head) {
   if (*head == NULL) {
      printf("List is empty. No element to delete.\n");
      return;
   }
   struct Node* temp = *head;
   *head = (*head)->next;
   free(temp);
```

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

// Function to delete a specified element
void deleteSpecified(struct Node** head, int element) {
    if (*head == NULL) {
        printf("List is empty. No element to delete.\n");
        return;
    }

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

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

    if (temp == NULL) {
        printf("Element %d not found.\n", element);
        return;
    }

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

// Function to delete the last element
void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. No element to delete.\n");
        return;
    }

    struct Node *temp = *head, *prev = NULL;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        printf("Last element deleted.\n");
        return;
    }
```

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

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

// Function to display the list
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

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

// Main function
int main() {
    struct Node* head = NULL;
    int choice, element;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Linked List\n");
        printf("2. Delete First Element\n");
        printf("3. Delete Specified Element\n");
        printf("4. Delete Last Element\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createList(&head);
                break;
```

```
            case 2:
                deleteFirst(&head);
                break;
            case 3:
                printf("Enter the element to delete: ");
                scanf("%d", &element);
                deleteSpecified(&head, element);
                break;
            case 4:
                deleteLast(&head);
                break;
            case 5:
                display(head);
                break;
            case 6:
                printf("Exiting the program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

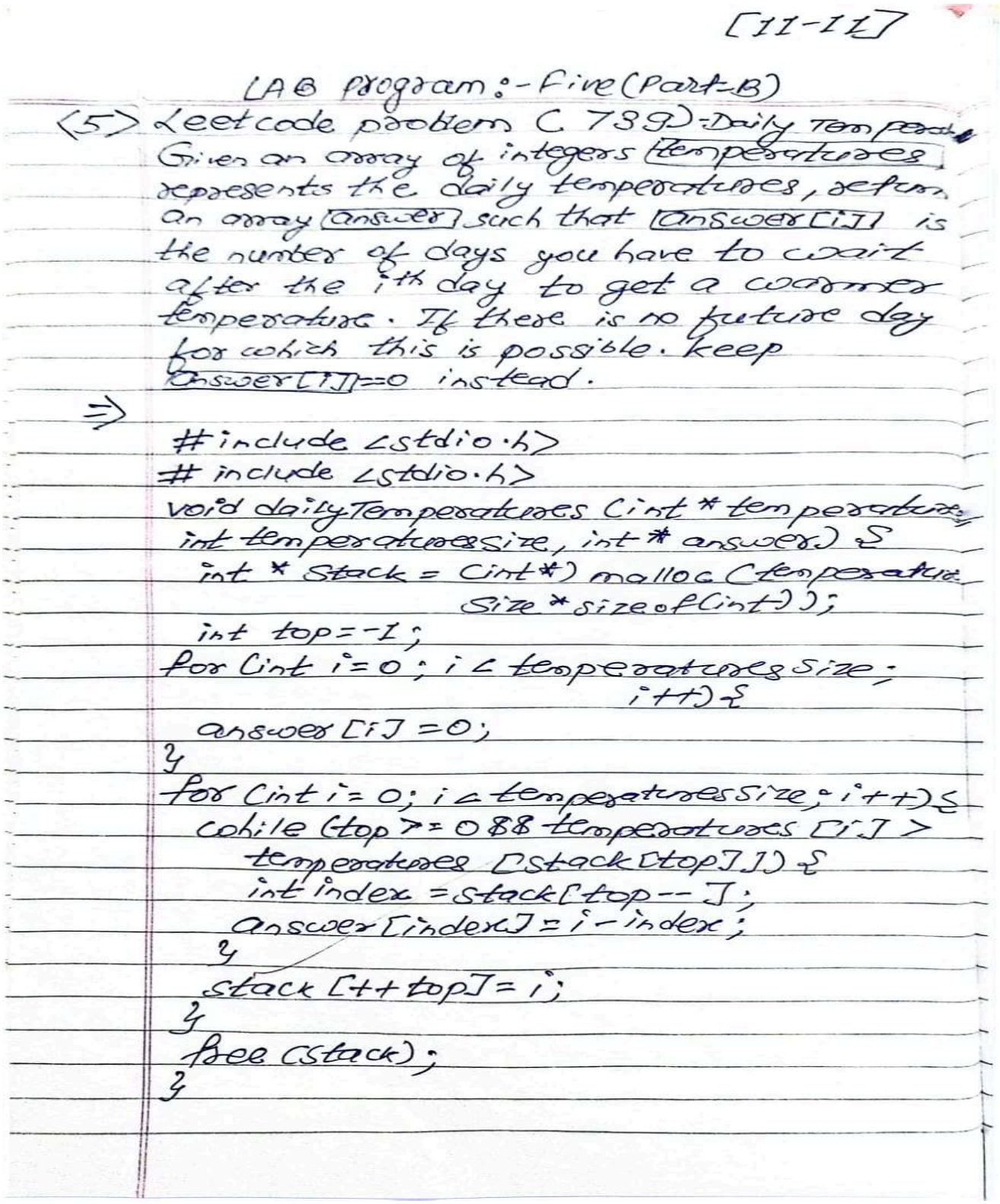
Output:

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the number of elements: 5
Enter element 1: 10
Enter element 2: 20
Enter element 3: 30
Enter element 4: 40
Enter element 5: 50

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

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List contents: 20 -> 30 -> 40 -> 50 -> NULL
```

```
Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 3
Enter the element to delete: 30
Element 30 deleted.

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List contents: 20 -> 40 -> 50 -> NULL

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

Menu:
1. Create Linked List
2. Delete First Element
3. Delete Specified Element
4. Delete Last Element
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List contents: 20 -> 40 -> NULL
```

B)Program - Leetcode platform (Daily Temperature).

Algorithm:



[II-II]

LAB program:- Five (Part-B)

(5) Leetcode problem (739) :- Daily Temperatures

Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the i^{th} day to get a warmer temperature. If there is no future day for which this is possible, keep answer[i]==0 instead.

⇒

```
#include <stdio.h>
#include <stdio.h>
void dailyTemperatures (int * temperatures,
 int temperaturesSize, int * answer) {
  int * stack = (int*) malloc (temperaturesSize * sizeof(int));
  int top=-1;
for (int i=0; i < temperaturesSize;
                                  i++) {
  answer [i] =0;
}
for (int i= 0; i < temperaturesSize; i++) {
  while (top >= 0 && temperatures [i] >
     temperatures [stack[top]]) {
     int index = stack[top--];
     answer[index] = i - index;
  }
  stack [++top] = i;
}
 free (stack);
}
```

```
int main(){
  int n;
  printf("Enter the number of days: ");
  scanf("%d", &n);
  int * temperatures = (int *) malloc
                    (n * sizeof(int));
  int * answer = (int *) malloc (n * sizeof
                    (int));
  printf("Enter the temperatures for
              each day: \n");
  for(int i = 0; i<n; i++){
     printf("Day %d: ", i+1);
     scanf("%d", &temperatures[i]);
  }
  dailyTemperatures(temperatures, n, answer);
  printf("Number of days to wait for a
          warmer temperature: \n");
  for(int i=0; i<n; i++){
     printf("%d ", answer[i]);
  }
  printf("\n");

  free(temperatures);
  free(answer);

  return 0;
}
```

Output:-

```
Enter the number of days: 6
Enter the temperatures for each day:
Day 1: 20
Day 2: 21
Day 3: 22
Day 4: 23
Day 5: 24
Day 6: 25
Number of days to wait for a warmer temperature:
1 1 1 1 1 0
```

Code:

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

void dailyTemperatures(int* temperatures, int temperaturesSize, int* answer) {
    int* stack = (int*)malloc(temperaturesSize * sizeof(int));
    int top = -1;

    for (int i = 0; i < temperaturesSize; i++) {
        answer[i] = 0;
    }

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

    free(stack);
}
int main() {
    int n;

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

    int* temperatures = (int*)malloc(n * sizeof(int));
    int* answer = (int*)malloc(n * sizeof(int));

    printf("Enter the temperatures for %d days: \n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &temperatures[i]);
    }
    dailyTemperatures(temperatures, n, answer);
    printf("Answer (Number of days to wait for warmer temperature): \n");
    for (int i = 0; i < n; i++) {
        printf("%d ", answer[i]);
    }
    printf("\n");

    free(temperatures);
    free(answer);

    return 0;
}
```

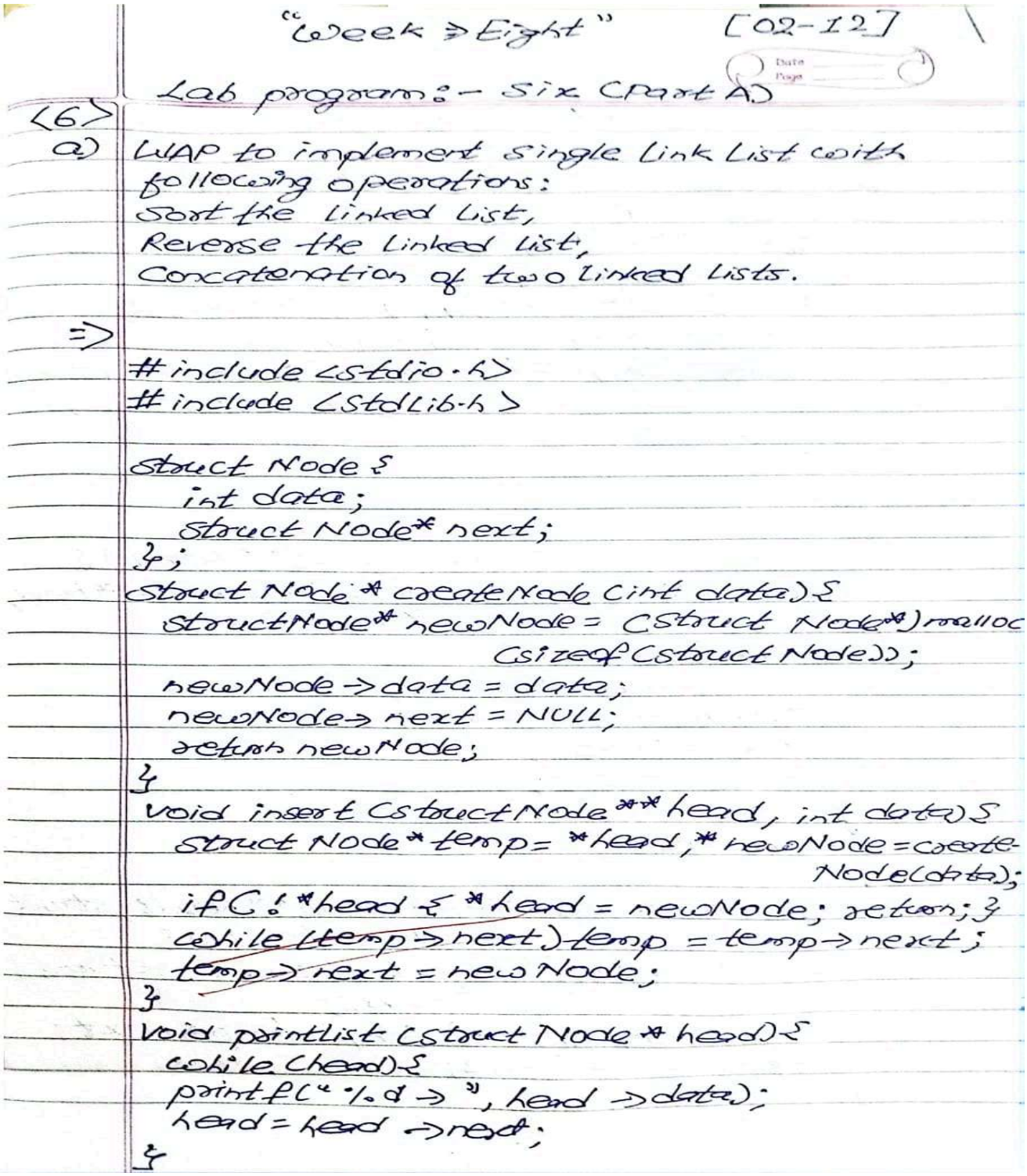
Output:

```
Enter the number of days: 6
Enter the temperatures for 6 days:
20
21
22
23
24
25
Answer (Number of days to wait for warmer temperature):
1 1 1 1 1 0
```

Program:6

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

Algorithm:



"Week ⇒ Eight" [02-12]

Lab program:- six (part A)

(6)

a) WAP to implement single link List with following operations:
Sort the linked list,
Reverse the linked list,
Concatenation of two linked lists.

⇒

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

struct Node {
    int data;
    struct Node* next;
};
struct Node * createNode (int data) {
    structNode* newNode = (struct Node*) malloc
                          (sizeof(struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
void insert (struct Node **head, int data) {
    struct Node * temp = *head, * newNode = create-
                                            Node(data);
    if(!*head) { *head = newNode; return; }
    while (temp -> next) temp = temp -> next;
    temp -> next = newNode;
}
void printlist (struct Node * head) {
    while (head) {
    printf("%d -> ", head -> data);
    head = head -> next;
}
```

```
    printf("NULL\n");
}
void sortList(struct Node* head){
    if(!head) return;
    struct Node *i, *j;
    for(i=head; i; i=i->next){
        for(j=i->next; j; j=j->next){
            if(i->data > j->data){
                int temp = i->data; i->data = j->data;
                                    j->data = temp;
            }
        }
    }
}
void reverseList(struct Node** head){
    struct Node *prev = NULL, *current = *head,
                              *next = NULL;
    while(current){
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}
void concatenate(struct Node** head1, struct
                                  Node* head2){
    if(!*head1){*head1 = head2; return;}
    struct Node* temp = *head1;
    while(temp->next) temp = temp->next;
    temp->next = head2;
}
```

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

printf("Enter number of elements in list1:");
scanf("%d", &n);
printf("Enter elements for list 1:");
while(n--) {scanf("%d", &data); insert
                    (&list1, data); }

printf("Enter number of elements in list2:");
scanf("%d", &m);
printf("Enter elements for list 2:");
while(m--) {scanf("%d", &data);
                    insert(&list2, data);}

printf("\nOriginal list1:");
printList(list1);
printf("Original list 2:");
printlist(list2);

sortlist(list1);
printf("Sorted list1:"); printlist(list1);

reverselist(&list1);
printf("Reversed list 1:"); printlist(list1);

concatenate(&list1, list2);
printf("Concatenated list:"); printlist(list1);

return 0;
}
```

Output:-

```
Enter number of elements in List1: 4
Enter elements for list1: 10 30 20 5
Enter number of elements in List2: 2
Enter elements for list2: 50 40

Original List1 : 10 -> 30 -> 20 -> 5 -> NULL
Original List2 : 50 -> 40 -> NULL
sorted List1 : 5 -> 10 -> 20 -> 30 -> NULL
Reversed List1 : 30 -> 20 -> 10 -> 5 -> NULL
Concatenated list : 30 -> 20 -> 10 -> 5 -> 50 ->
                    40 -> NULL
```

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

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

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

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

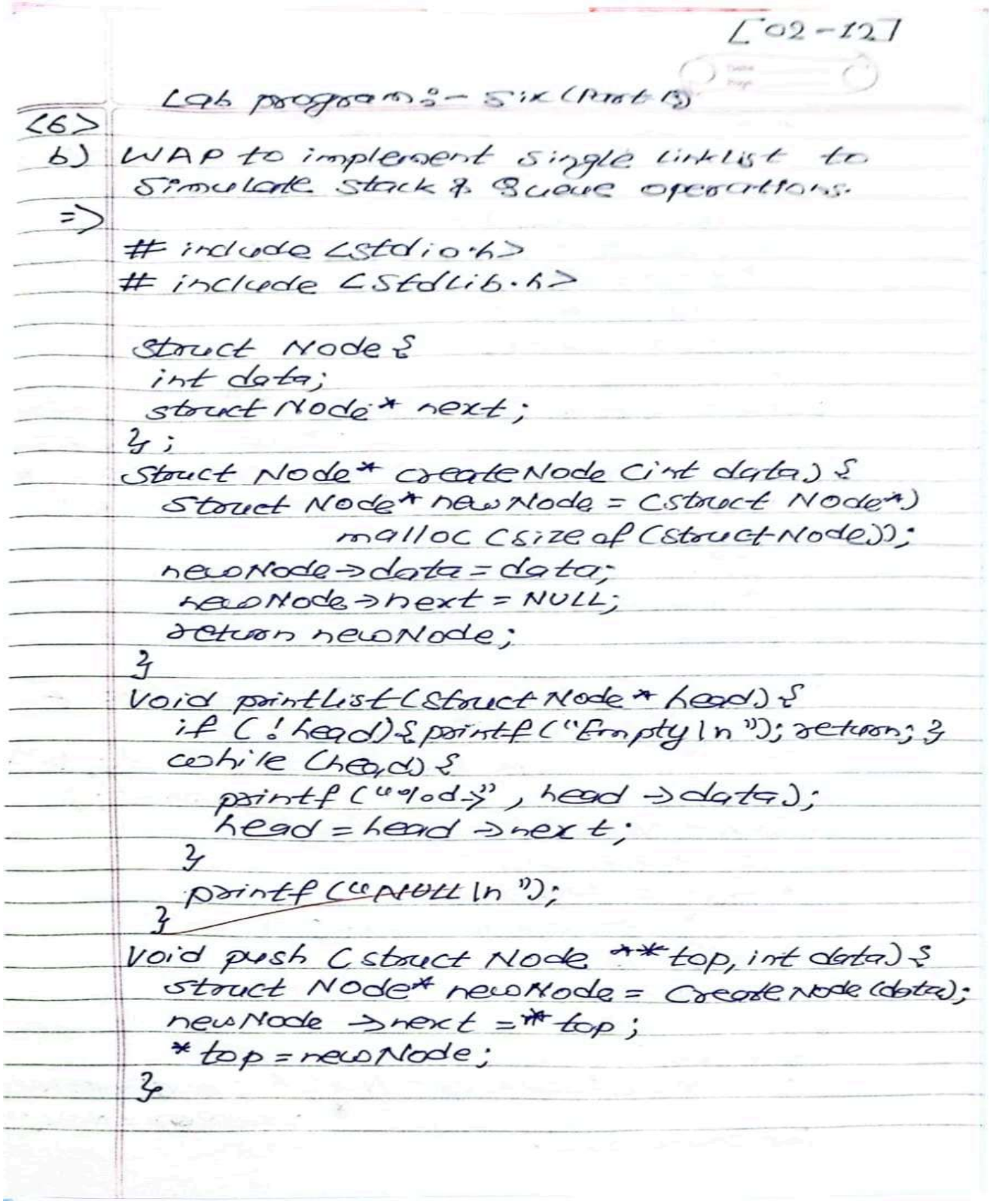
```
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

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

int main() {
    struct Node *list1 = NULL, *list2 = NULL;
    int n, m, data;

    printf("Enter number of elements in List 1: ");
    scanf("%d", &n);
    printf("Enter elements for List 1: ");
    while (n--) { scanf("%d", &data); insert(&list1, data); }

    printf("Enter number of elements in List 2: ");
    scanf("%d", &m);
    printf("Enter elements for List 2: ");
    while (m--) { scanf("%d", &data); insert(&list2, data); }

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

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

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

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

    return 0;
}
```

Output:

```
Enter number of elements in List 1: 4
Enter elements for List 1: 10 30 20 5
Enter number of elements in List 2: 2
Enter elements for List 2: 50 40

Original List 1: 10 -> 30 -> 20 -> 5 -> NULL
Original List 2: 50 -> 40 -> NULL
Sorted List 1: 5 -> 10 -> 20 -> 30 -> NULL
Reversed List 1: 30 -> 20 -> 10 -> 5 -> NULL
Concatenated List: 30 -> 20 -> 10 -> 5 -> 50 -> 40 -> NULL
```

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

Algorithm:



```
[02-12]

Lab program 5 - Six (Part B)

(6)
b) WAP to implement single linklist to
simulate stack & queue operations.
=>

#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 printList(struct Node* head) {
  if(!head){printf("Empty\n"); return;}
  while(head) {
     printf("%d->", head->data);
     head = head->next;
  }
  printf("NULL\n");
}
Void push(struct Node **top, int data) {
  struct Node* newNode = createNode(data);
  newNode->next = *top;
  *top = newNode;
}
```

```
int pop (struct Node** top){
  if (!*top) {printf("Stack empty!\n");
                    return -1;}
  struct Node *temp = *top;
  int data = temp->data;
  *top = temp->next;
  free(temp);
  return data;
}
void enqueue (struct Node** front, struct
                Node** rear, int data){
  struct Node * newNode = createNode(data);
  if(!*rear) *front = *rear = newNode;
  else{
    (*rear)->next = newNode;
    *rear = newNode;
  }
}
int dequeue (struct Node** front, struct
              Node** rear){
  if(!*front) {printf("Queue empty!\n");
                    return -1;}
  struct Node *temp = *front;
  int data = temp->data;
  *front = temp->next;
  if(!*front) *rear = NULL;
  free(temp);
  return data;
}
int main(){
  struct Node *stackTop = NULL, *queueFront = NU
                          *queueRear = NULL;
  int choice, data;
```

```
while(1){
  printf("\n1. Push to stack\n2. Pop from
stack\n3. Enqueue to queue\n4. Dequeue
from queue\n5. Print stack\n6. Print queue\n
7. Exit\n Enter choice: ");
scanf("%d", &choice);

Switch (choice){
 case 1:
    printf("Enter data to push: ");
    scanf("%d", &data);
    push (&stackTop, data);
    break;
case 2:
    data = pop(&stackTop);
    if(data != -1) printf("popped from
                stack: %d\n", data);
    break;
case 3:
    printf("Enter data to enqueue: ");
    scanf("%d", &data);
    enqueue(&queueFront, &queueRear,
                              data);
    break;
case 4:
    data = dequeue(&queueFront, &queue
                                Rear);
    if(data != -1) printf("Dequeued from
                queue: %d\n", data);
    break;
case 5:
    printf("stack: "); printList(stackTop);
    break;
```

```
case 6:
    printf("Queue: "); printlist(queueFront);
    break;
case 7:
    exit(0);
default:
    printf("Invalid choice! \n");
  }
}
return 0;
}
```

Output :-

```
1. push to stack
2. pop from stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. print Queue
7. Exit
Enter choice : 1
Enter data to push : 10

1. push to stack
2. pop from stack
3. Enqueue to Queue
4. Dequeue from Queue
5. print stack
6. print Queue
7. Exit
```

Enter choice: 3
Enter data to enqueue: 20

1. push to stack
2. pop from stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print stack
6. Print Queue
7. Exit
Enter choice: 5
Stack: 10 -> NULL

Enter choice: 6
Queue: 20 -> NULL

Enter choice: 4
Dequeued from Queue: 20

Seen
02/12/24
Enter choice: 5
Stack: 10 -> NULL
~~Queue: Empty~~
Enter choice: 6
Queue: Empty

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 printList(struct Node* head) {
   if (!head) { printf("Empty\n"); return; }
   while (head) {
      printf("%d -> ", head->data);
      head = head->next;
   }
   printf("NULL\n");
}
void push(struct Node** top, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *top;
   *top = newNode;
}
int pop(struct Node** top) {
   if (!*top) { printf("Stack empty!\n"); return -1; }
   struct Node* temp = *top;
   int data = temp->data;
   *top = temp->next;
   free(temp);
   return data;
}
void enqueue(struct Node** front, struct Node** rear, int data) {
   struct Node* newNode = createNode(data);
   if (!*rear) *front = *rear = newNode;
   else {
      (*rear)->next = newNode;
      *rear = newNode;
   }
}

int dequeue(struct Node** front, struct Node** rear) {
   if (!*front) { printf("Queue empty!\n"); return -1; }
```

```
    struct Node* temp = *front;
    int data = temp->data;
    *front = temp->next;
    if (!*front) *rear = NULL;
    free(temp);
    return data;
}
int main() {
    struct Node *stackTop = NULL, *queueFront = NULL, *queueRear = NULL;
    int choice, data;
    while (1) {
        printf("\n1. Push to Stack\n2. Pop from Stack\n3. Enqueue to Queue\n4. Dequeue from
Queue\n5. Print Stack\n6. Print Queue\n7. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                push(&stackTop, data);
                break;
            case 2:
                data = pop(&stackTop);
                if (data != -1) printf("Popped from Stack: %d\n", data);
                break;
            case 3:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&queueFront, &queueRear, data);
                break;
            case 4:
                data = dequeue(&queueFront, &queueRear);
                if (data != -1) printf("Dequeued from Queue: %d\n", data);
                break;
            case 5:
                printf("Stack: "); printList(stackTop);
                break;
            case 6:
                printf("Queue: "); printList(queueFront);
                break;
            case 7:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

Output:

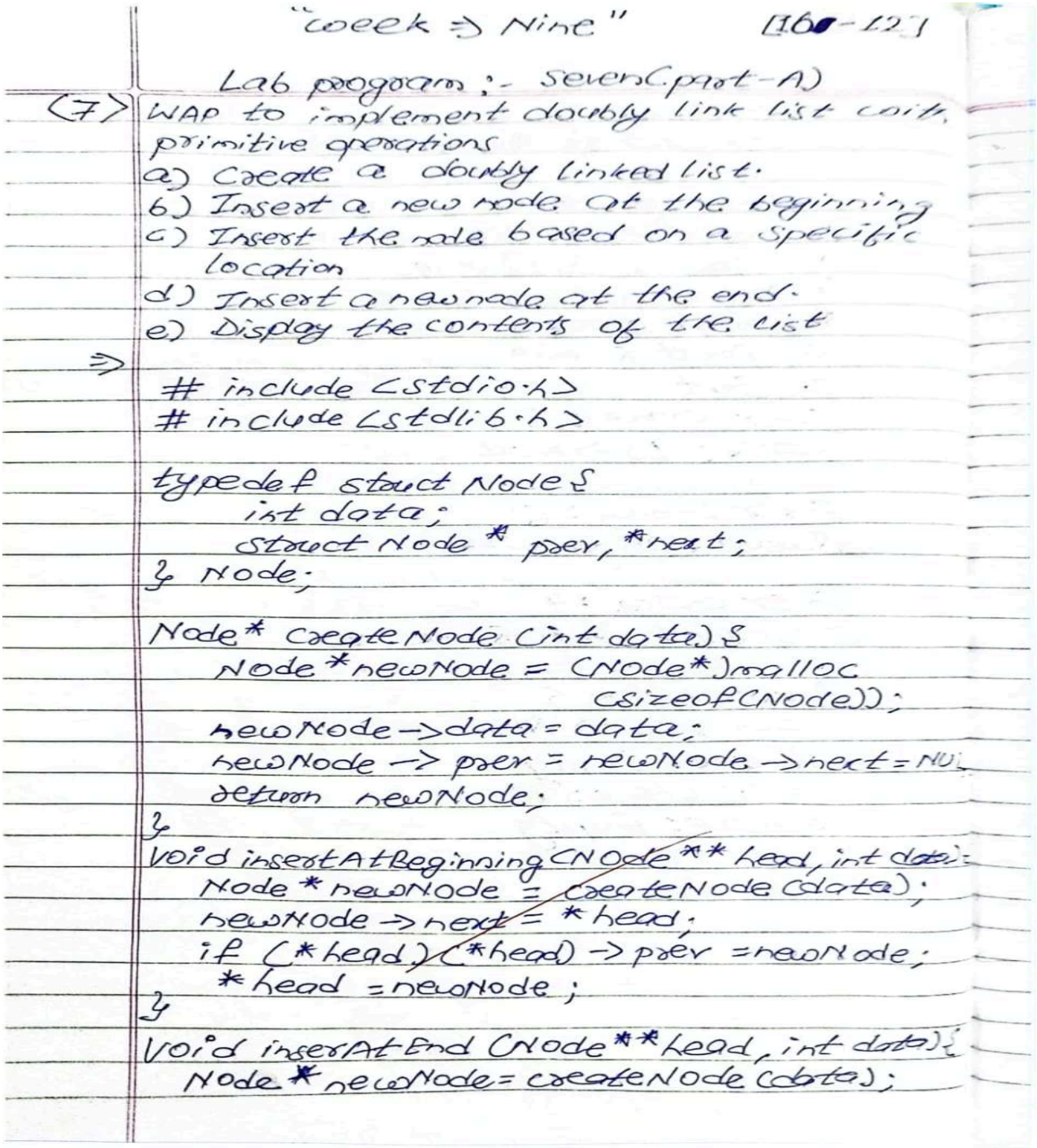
```
1. Push to Stack
2. Pop from Stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. Print Queue
7. Exit
Enter choice: 1
Enter data to push: 10

1. Push to Stack
2. Pop from Stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. Print Queue
7. Exit
Enter choice: 3
Enter data to enqueue: 20

1. Push to Stack
2. Pop from Stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. Print Queue
7. Exit
Enter choice: 5
Stack: 10 -> NULL
```

```
1. Push to Stack
2. Pop from Stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. Print Queue
7. Exit
Enter choice: 6
Queue: 20 -> NULL

1. Push to Stack
2. Pop from Stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. Print Queue
7. Exit
Enter choice: 4
Dequeued from Queue: 20

1. Push to Stack
2. Pop from Stack
3. Enqueue to Queue
4. Dequeue from Queue
5. Print Stack
6. Print Queue
7. Exit
Enter choice: 5
Stack: 10 -> NULL

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

Program 7:

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 .

Algorithm:



"week => Nine" [16B-12]

Lab program :- seven(part-A)

(7) 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

=>

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

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

Node* createNode (int data) {
    Node *newNode = (Node*)malloc
                    (sizeof(Node));
    newNode->data = data;
    newNode -> prev = newNode -> next = NU
    return newNode;
}
void insertAtBeginning(Node ** head, int data
    Node *newNode = createNode(data);
    newNode -> next = *head;
    if (*head) (*head) -> prev = newNode;
    *head = newNode;
}
Void insertAtEnd (Node **head, int data){
  Node *newNode = createNode(data);
```

```
    if (*head == NULL) *head = newNode;
    else {
        Node *temp = *head;
        while (temp->next) temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }
}
void insertAtPosition(Node **head, int data,
                      int pos){
    if (pos == 1) return insertAtBeginning
                         (head, data);
    Node *newNode = createNode(data);
    Node *temp = *head;
    for (int i = 1; temp && i < pos-1; i++)
        temp = temp->next;
    if (temp == NULL) return;
    newNode->next = temp->next;
    if (temp->next) temp->next->prev =
                                newNode;
    temp->next = newNode;
    newNode->prev = temp;

void displayCNode *head) {
    while (head) {
        printf("%d", head->data);
        head = head->next;
    }
    printf("\n");
```

```
int main(){
    Node *head = NULL;
    int choice, data, position;
    while(1){
        printf("\n1. Insert at Beginning
\n2. Insert at End \n3. Insert at position.
\n4. Display \n5. Exit \n");
        printf("Enter choice : ");
        scanf("%d", &choice);

    switch(choice){
    case 1:
        printf("Enter data:");
        scanf("%d", &data);
        insertAtBeginning(&head, data);
        break;
    case 2:
        printf("Enter data: ");
        scanf("%d", &data);
        insertAtEnd(&head, data);
        break;
    case 3:
        printf("Enter data and position: ");
        scanf("%d %d", &data, &position);
        insertAtposition(&head, data, position);
        break;
    case 4:
        display(head);
        break;
    case 5:
        return 0;
    default
        print("Invalid choice\n");
    }
}
```

Output:-

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data: 10

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data: 20

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data: 30

1. Insert at Beginning
2. Insert at End
3. Insert at position

4. Display
5. Exit
Enter choice: 3
Enter data and position: 25 2

.

1. Insert at beginning
2. Insert at End
3. Insert at position
4. Display
5. Exit
Enter choice: 4
30 25 20 10

1. Insert at Beginning
2. Insert at End
3. Insert at position
4. Display
5. Exit
Enter choice: 5
Exit

Seen
16/12/24

Code:

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

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

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

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

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

void insertAtPosition(Node **head, int data, int pos) {
   if (pos == 1) return insertAtBeginning(head, data);
   Node *newNode = createNode(data);
   Node *temp = *head;
   for (int i = 1; temp && i < pos - 1; i++) temp = temp->next;
   if (temp == NULL) return;  // Invalid position
   newNode->next = temp->next;
   if (temp->next) temp->next->prev = newNode;
   temp->next = newNode;
   newNode->prev = temp;
}

void display(Node *head) {
```

```
    while (head) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

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

    while (1) {
        printf("\n1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n5.
Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 3:
                printf("Enter data and position: ");
                scanf("%d %d", &data, &position);
                insertAtPosition(&head, data, position);
                break;
            case 4:
                display(head);
                break;
            case 5:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
}
```

Output:

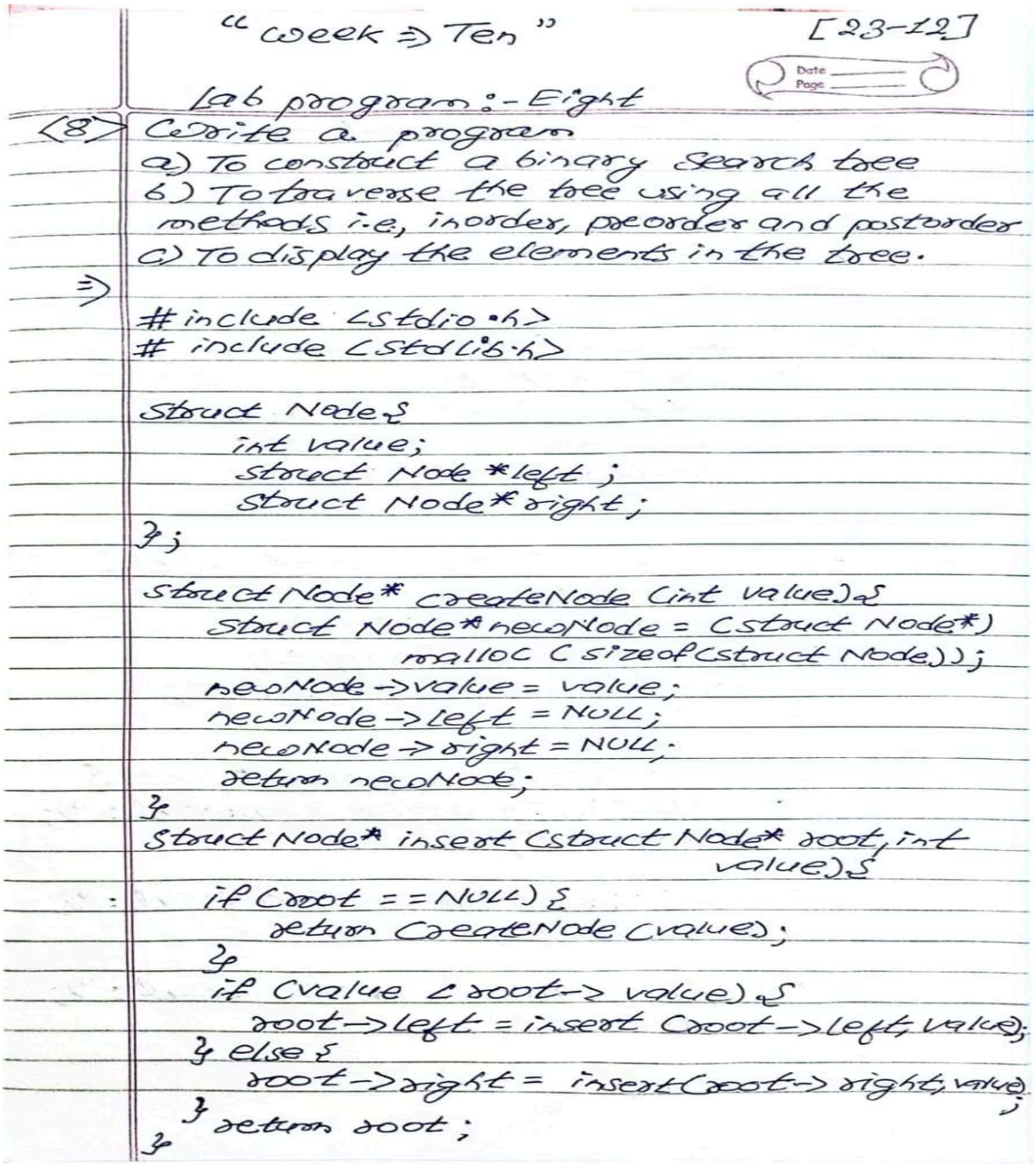
```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter data: 10

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter data: 20

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter data: 30
```

```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 3
Enter data and position: 25 2

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 4
30 25 20 10

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 5
```

Program 8:

Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in order, preorder and post order c) To display the elements in the tree.

Algorithm:



"Week ⇒ Ten" [23-12]

Lab program :- Eight

(8) Write a program
a) To construct a binary search tree
b) To traverse the tree using all the methods i.e, inorder, preorder and postorder
c) To display the elements in the tree.

⇒

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

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

struct Node* createNode(int value){
    struct Node* newNode = (struct Node*)
            malloc(sizeof(struct Node));
    newNode->value = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct Node* insert(struct Node* root, int
                                value){
    if(root == NULL){
        return createNode(value);
    }
    if (value < root->value){
        root->left = insert(root->left, value);
    } else{
        root->right = insert(root->right, value);
    }
    return root;
}
```

```
void inorder(struct Node* root){
    if(root != NULL){
        inorder(root->left);
        printf("%d ", root->value);
        inorder(root->right);
    }
}
void preorder(struct Node* root){
    if(root != NULL){
        printf("%d", root->value);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct Node* root){
    if(root != NULL){
        postorder(root->left);
        postorder(root->right);
        printf("%d", root->value);
    }
}
void displayTree(struct Node* root){
    printf("\n In-order Traversal: ");
    inorder(root);

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

    printf("\n post-order Traversal: ");
    postorder(root);
}
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 of the tree:\n");
for(int i=0; i<n; i++){
    scanf("%d", &value);
    root=insert(root, value);
}
displayTree(root);
return 0;
}
```

Output:-

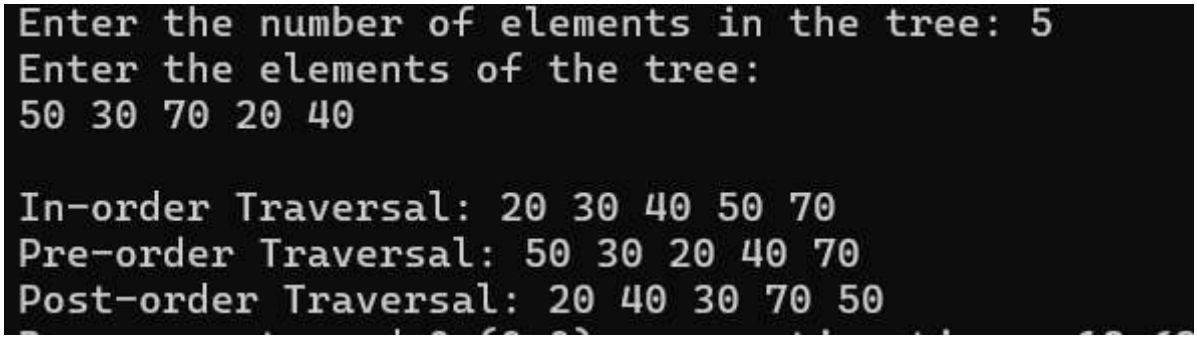
```
Enter the number of elements in the tree: 5
Enter the elements of the tree:
50 30 70 20 40

In-order Traversal: 20 30 40 50 70
pre-order Traversal: 50 30 20 40 70
Post-order Traversal: 20 40 30 70 50
```

Code:

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

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

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

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

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

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

void postorder(struct Node* root) {
```

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

void displayTree(struct Node* root) {
    printf("\nIn-order Traversal: ");
    inorder(root);

    printf("\nPre-order Traversal: ");
    preorder(root);

    printf("\nPost-order Traversal: ");
    postorder(root);
}

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

    displayTree(root);

    return 0;
}
```

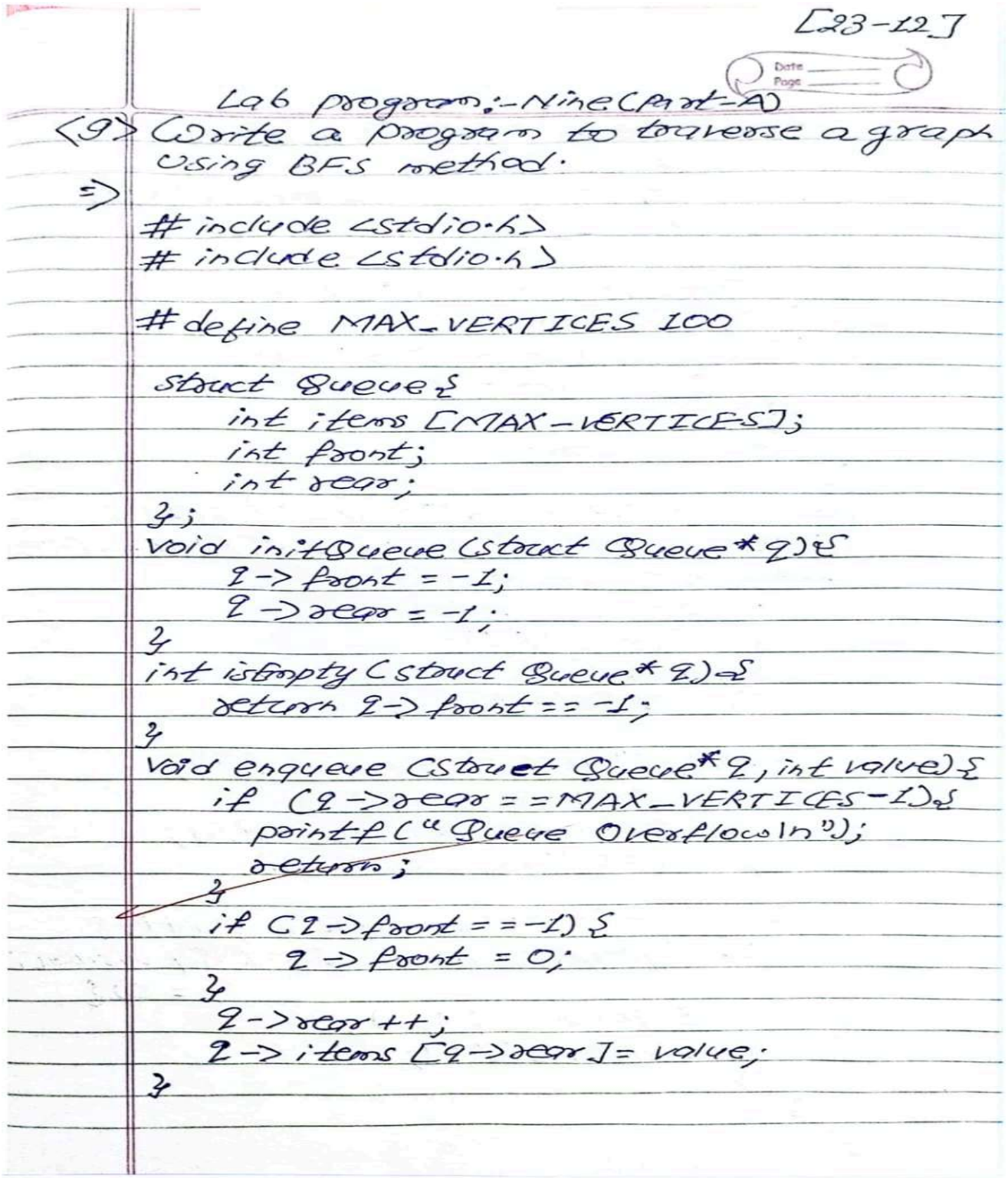
Output:


Enter the number of elements in the tree: 5
Enter the elements of the tree:
50 30 70 20 40
In-order Traversal: 20 30 40 50 70
Pre-order Traversal: 50 30 20 40 70
Post-order Traversal: 20 40 30 70 50

Program 9:

A) Write a program to traverse a graph using BFS method.

Algorithm:



```
[23-12-]

Lab program:- Nine(Part-A)

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

=>

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

#define MAX_VERTICES 100

struct Queue{
    int items [MAX-VERTICES];
    int front;
    int rear;
};
void initQueue (struct Queue* q){
    q -> front = -1;
    q -> rear = -1;
}
int isEmpty (struct Queue* q){
    return q -> front == -1;
}
void enqueue (struct Queue* q, int value){
    if (q -> rear == MAX_VERTICES-1){
        printf("Queue Overflow\n");
        return;
    }
    if (q -> front == -1){
        q -> front = 0;
    }
    q -> rear++;
    q -> items [q -> rear] = value;
}
```

```
int dequeue (struct Queue* q) {
    if (isEmpty (q)) {
        printf ("Queue Underflow\n");
        return -1;
    }
    int item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}
void BFS (int graph [MAX_VERTICES
[MAX_VERTICES], int visited [MAX_VERTICES
int start, int vertices) {
    struct Queue q;
    initQueue (&q);
    enqueue (&q, start);
    visited [start] = 1;

    printf ("BFS Traversal : ");
    while (!isEmpty (&q)) {
        int current = dequeue (&q);
        printf ("%d ", current);

        for (int i = 0; i < vertices ; i++) {
            if (graph [current][i] == 1 && visited[i]
                                        == 0) {
                enqueue (&q, i);
                visited [i] = 1;
            }
        }
    }
```

```
    printf("\n");
}
int main(){
    int vertices, edges, src, dest;

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

    int graph[MAX_VERTICES][MAX_VERTICES]
                                    = {0};
    int visited[MAX_VERTICES] = {0};

    printf("Enter the edges (source destination)
                                    :\n");
    for(int i=0; i<edges; i++){
        scanf("%d %d", &src, &dest);
        graph[src][dest] = 1;
        graph[dest][src] = 1;
    }
    int start;
    printf("Enter the starting vertex for
                                    BFS: ");
    scanf("%d", &start);

    BFS(graph, visited, start, vertices);

    return 0;
}
```

Output :-

```
Enter the number of vertices : 5
Enter the number of edges : 6
Enter the edges (source destination) :
0 1
0 2
1 3
1 4
2 4
3 4
Enter the starting vertex for BFS : 0
BFS Traversal : 0 1 2 3 4
```

Code:

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

#define MAX_VERTICES 100

struct Queue {
   int items[MAX_VERTICES];
   int front;
   int rear;
};

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

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

void enqueue(struct Queue* q, int value) {
   if (q->rear == MAX_VERTICES - 1) {
      printf("Queue Overflow\n");
      return;
   }
   if (q->front == -1) {
      q->front = 0;
   }
   q->rear++;
   q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
   if (isEmpty(q)) {
      printf("Queue Underflow\n");
      return -1;
   }
   int item = q->items[q->front];
   q->front++;
   if (q->front > q->rear) {
      q->front = q->rear = -1;
   }
   return item;
}

void BFS(int graph[MAX_VERTICES][MAX_VERTICES], int visited[MAX_VERTICES], int
```

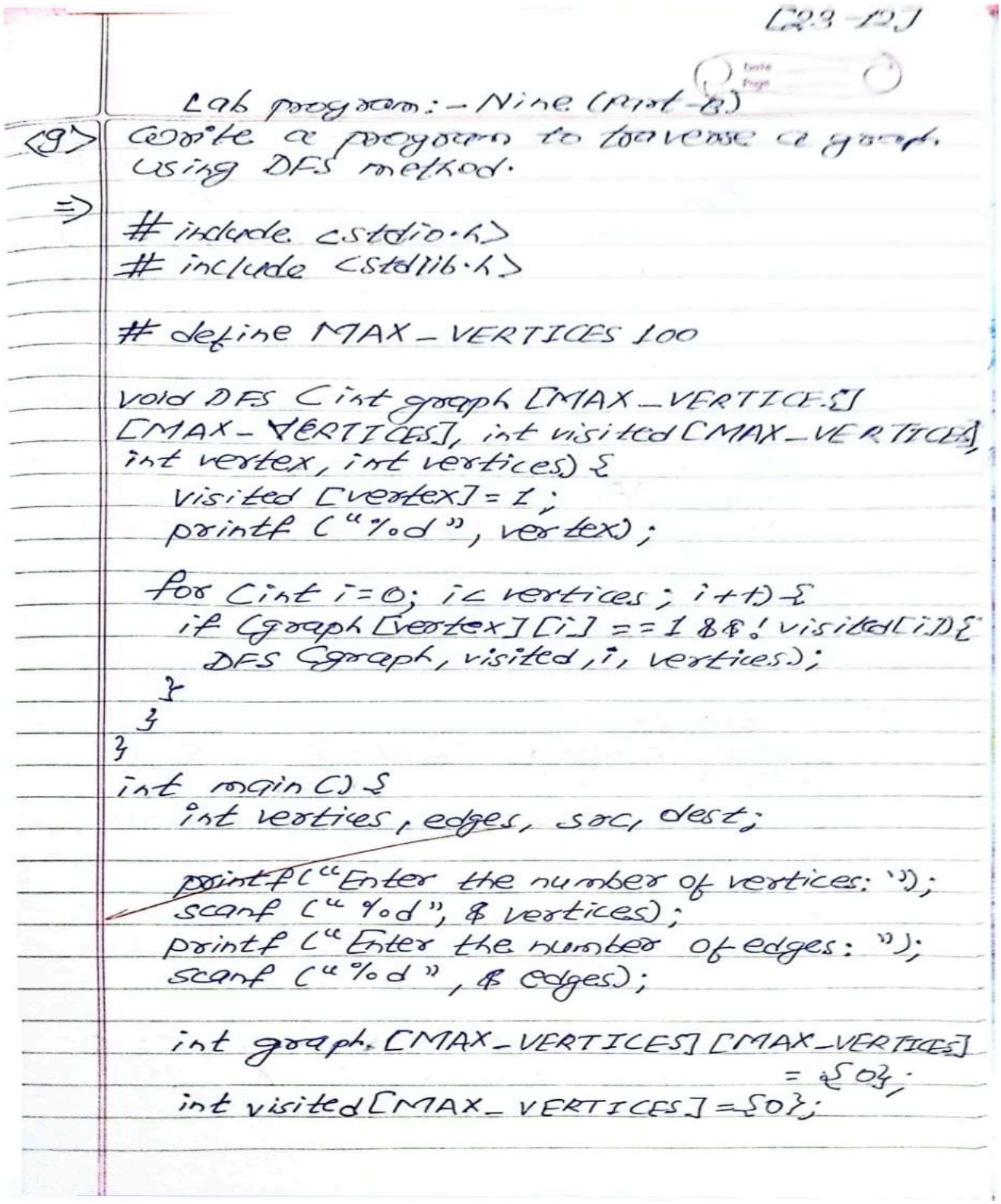
```
start, int vertices) {
    struct Queue q;
    initQueue(&q);
    enqueue(&q, start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (!isEmpty(&q)) {
        int current = dequeue(&q);
        printf("%d ", current);

        for (int i = 0; i < vertices; i++) {
            if (graph[current][i] == 1 && visited[i] == 0) {
                enqueue(&q, i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

int main() {
    int vertices, edges, src, dest;

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

    int graph[MAX_VERTICES][MAX_VERTICES] = {0};
    int visited[MAX_VERTICES] = {0};

    printf("Enter the edges (source destination):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &src, &dest);
        graph[src][dest] = 1;
        graph[dest][src] = 1;
    }

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

    BFS(graph, visited, start, vertices);

    return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (source destination):
0 1
0 2
1 3
1 4
2 4
3 4
Enter the starting vertex for BFS: 0
BFS Traversal: 0 1 2 3 4
```

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

Algorithm:



Lab program:- Nine (Part-B)

(9) Write a program to traverse a graph using DFS method.

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

#define MAX_VERTICES 100

void DFS (int graph [MAX_VERTICES]
[MAX_VERTICES], int visited [MAX_VERTICES],
int vertex, int vertices) {
    visited [vertex] = 1;
    printf ("%d", vertex);

    for (int i=0; i< vertices ; i++) {
      if (graph [vertex] [i] == 1 && ! visited [i]) {
        DFS (graph, visited, i, vertices);
      }
    }
}
int main () {
    int vertices, edges, src, dest;

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

    int graph [MAX_VERTICES] [MAX_VERTICES]
                                = {0};
    int visited [MAX_VERTICES] = {0};
```

```
    printf("Enter the edges (source destination):
                                        \n");
    for (int i=0; i<edges; i++){
        scanf("%d %d", &src, &dest);
    graph [src] [dest] = 1;
    graph [dest] [src] = 1;
}
int start;
printf ("Enter the starting vertex for
                                    DFS: ");
scanf ("%d", &start);
printf("DFS Traversal: ");
DFS (graph, visited, start, vertices);
printf ("\n");

return 0;
}
```

Output:-

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (source destination):
0 1
0 2
1 3
1 4
2 4
3 4
Enter the starting vertix for DFS: 0
DFS Traversal: 0 1 3 4 2
```

Seen

23/12/24

Code:

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

#define MAX_VERTICES 100

void DFS(int graph[MAX_VERTICES][MAX_VERTICES], int visited[MAX_VERTICES], int
vertex, int vertices) {
   visited[vertex] = 1;
   printf("%d ", vertex);

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

int main() {
   int vertices, edges, src, dest;

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

   int graph[MAX_VERTICES][MAX_VERTICES] = {0};
   int visited[MAX_VERTICES] = {0};

   printf("Enter the edges (source destination):\n");
   for (int i = 0; i < edges; i++) {
      scanf("%d %d", &src, &dest);
      graph[src][dest] = 1;
      graph[dest][src] = 1;
   }

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

   printf("DFS Traversal: ");
   DFS(graph, visited, start, vertices);
   printf("\n");

   return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges (source destination):
0 1
0 2
1 3
1 4
2 4
3 4
Enter the starting vertex for DFS: 0
DFS Traversal: 0 1 3 4 2
```