VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SATISH GIRISH KUDARE (1BM23CS306)

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 **SATISH GIRISH KUDARE (1BM23CS306),** 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 academicrequirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

Lab faculty Incharge Geetha N Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link:
https://github.com/Satish1895/SATISH_DST_1BM23CS306

Program 1

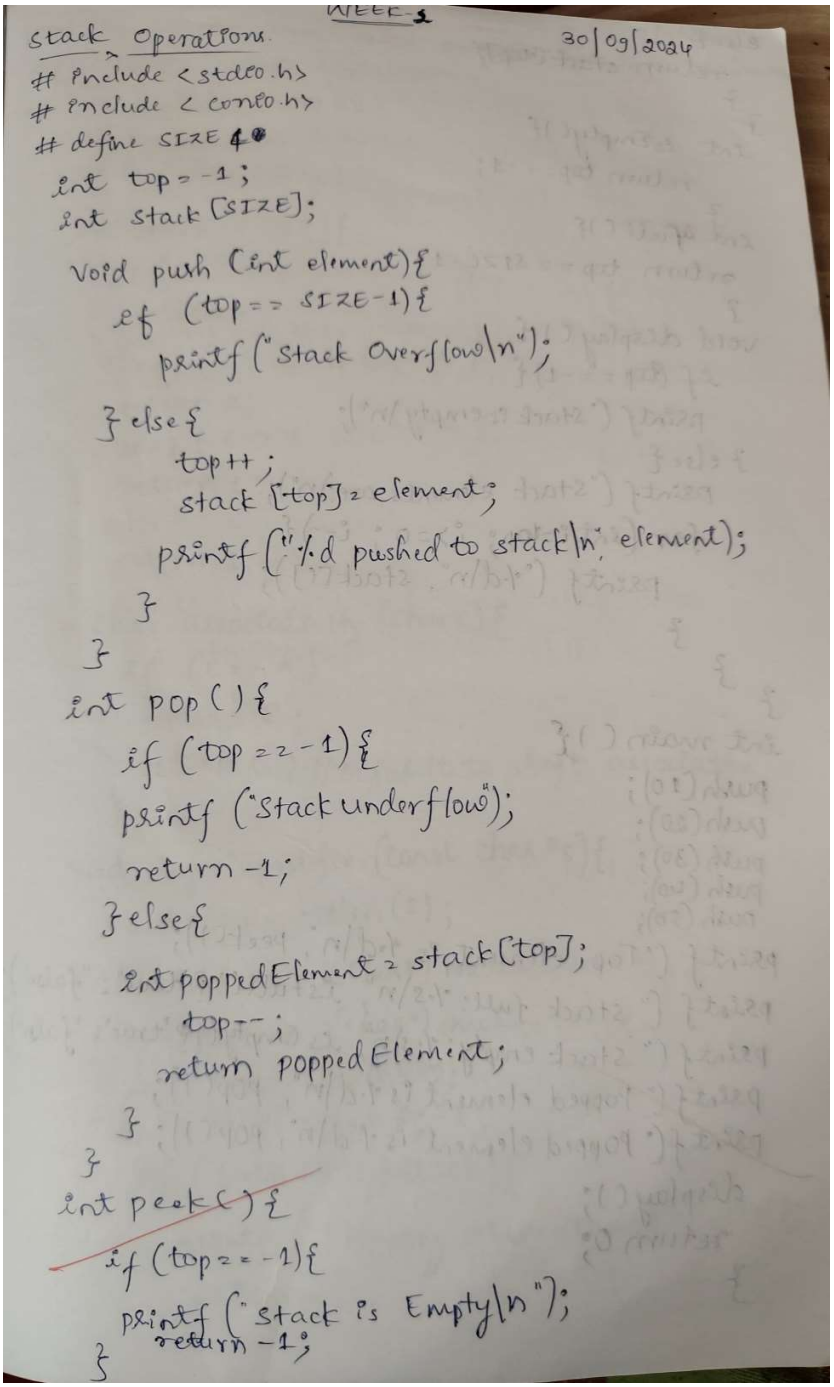
Write a program to simulate the working of stack using an array with the following: a) Push

b) Pop

c) Display

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

Observation:



```
WEEK 2
Stack Operations                                30/09/2024
#include <stdio.h>
#include <conio.h>
#define SIZE 4
int top = -1;
int stack[SIZE];

void push(int element){
    if (top == SIZE-1){
        printf("Stack Overflow\n");
    }else{
        top++;
        stack[top] = element;
        printf("%d pushed to stack\n", element);
    }
}
int pop(){
    if (top == -1){
    printf("Stack underflow");
    return -1;
    }else{
    int poppedElement = stack[top];
        top--;
        return poppedElement;
    }
}
int peek(){
    if (top == -1){
    printf("Stack is Empty\n");
        return -1;
}
```

```
else{
    return stack[top];
  }
}
int isEmpty(){
    return top==-1;
  }
int isfull(){
  return top==SIZE-1;
  }
void display(){
    if(top==-1){
    printf("stack is empty\n");
  }else{
    printf("stack elements are:\n");
    for(int i=top; i>=0; i--){
        printf("%d\n", stack[i]);
      }
  }
}
int main(){
 push(10);
 push(20);
 push(30);
 push(40);
 push(50);
printf("Top element is %d\n", peek());
printf("stack full: %s\n", isFull()?"true":"false");
printf("stack empty: %s\n", isEmpty()?"true":"false");
printf("Popped element is %d\n", pop());
printf("Popped element is %d\n", pop());
 display();
  return 0;
}
```

Code:

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

int top = -1;


int isEmpty(int arr[]) {
    return top == -1;
}

int isFull(int arr[], int limit) {
    return top == limit - 1;
}

int Top(int arr[]) {
    if (isEmpty(arr)) {
        printf("Stack is empty.\n");
        return -1;
    }
    return arr[top];
}

void Display(int arr[]) {
    if (isEmpty(arr)) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = top; i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}


void Push(int value, int arr[], int limit) {
    if (isFull(arr, limit)) {
        printf("Stack is full.\n");
    } else {
        top++;
        arr[top] = value;
        printf("Pushed %d onto the stack.\n", value);
    }
}

void Pop(int arr[]) {
    if (isEmpty(arr)) {
        printf("Stack is empty.\n");
    } else {
        printf("Popped %d from the stack.\n", arr[top]);
```

```
        top--;
    }
}

int main() {
    int limit;

    printf("Enter the limit of the stack: ");
    scanf("%d", &limit);

    int arr[limit];

    while (1) {
        int choice, value;
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Top\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                Push(value, arr, limit);
                break;
            case 2:
                Pop(arr);
                break;
            case 3:
                printf("Top element is: %d\n", Top(arr));
                break;
            case 4:
                Display(arr);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output:

```
PS C:\Users\satis>  & 'c:\Users\satis\.vscode\extensions\ms-vscode.cpptools-1.22.11-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-mljl4g2o.ym5' '--stdout=Microsoft-MIEngine-Out-0hcv5fjb.eug' '--stderr=Microsoft-MIEngine-Error-ewbwcvr2.gjn' '--pid=Microsoft-MIEngine-Pid-2xpmhtiz.bpy' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Enter the limit of the stack: 5

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

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

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

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

```
Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 2
Popped 5 from the stack.

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

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

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

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

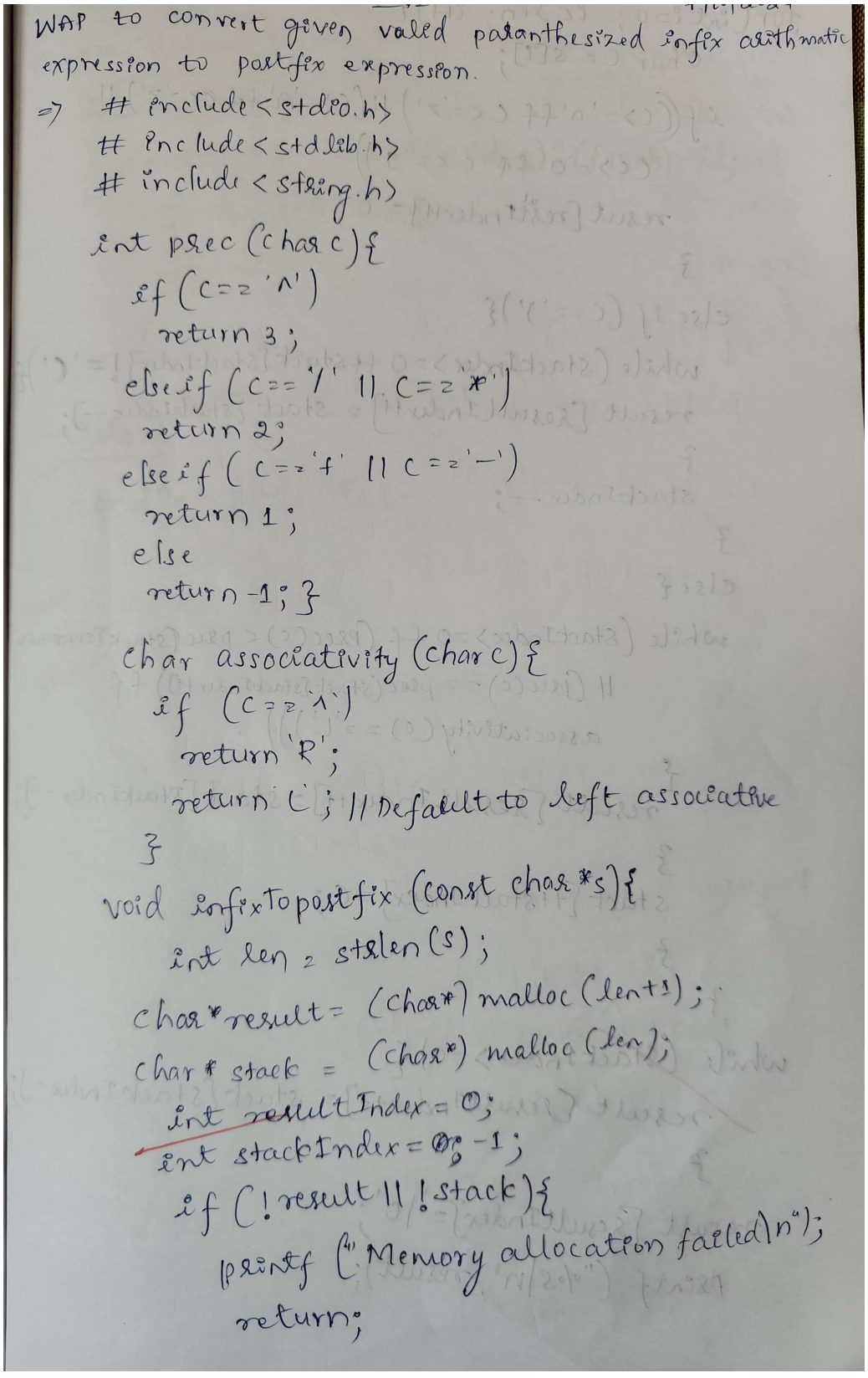
```
Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 5
Pushed 5 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 6
Stack is full.

Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 3
Top element is: 5

Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 4
Stack elements: 5 4 3 2 1
```

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

Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 2
Stack is empty.

Stack Operations:
1. Push
2. Pop
3. Top
4. Display
5. Exit
Enter your choice: 5
```

Program 2

Write a program to convert a given valid parenthesized in-fix arithmetic expression to post-fix expression.The expression consists of single character operands and the binary operators +,-,*,/.

Observation:



```
WAP to convert given valid paranthesized infix arithmatic
expression to postfix expression.
=> #include<stdio.h>
   #include<stdlib.h>
   #include<string.h>
   int prec(char c){
     if(c=='^')
       return 3;
     else if(c=='/'||c=='*')
       return 2;
     else if(c=='+'||c=='-')
       return 1;
     else
       return -1;}
   char associativity(char c){
     if(c=='^')
       return 'R';
       return 'L'; // Default to left associative
     }
   void infixtopostfix(const char *s){
       int len = strlen(s);
     char*result=(char*)malloc(len+1);
     char* stack = (char*) malloc(len);
       int resultIndex=0;
       int stackIndex=-1;
       if(!result||!stack){
         printf("Memory allocation failed\n");
         return;
```

```
for(int i=0; i<len; i++){
    char c = s[i];
    if((c>='a' && c<='z') || (c>='A' && c<='Z') ||
       (c>='0' && c<='9'))
        result[resultIndex++] = c;
    }
    else if (c==')'){
    while (stackIndex>=0 && stack[stackIndex]!='('){
    result [resultIndex++] = stack [stackIndex--];
    }
    stackIndex--;
}
else{
while (stackIndex>=0 && (prec(c) < prec(stack[stackIndex]
        || (prec(c)==prec(stack[stackIndex+1]) &&
        associativity(c)=='L')))
    {
    result [resultIndex++] = stack[stackIndex--];
    }
    stack[++stackIndex]=c;
    }
}
while (stackIndex>=0){
    result [resultIndex++]= stack[stackIndex--];
    }
result [resultIndex]='\0';
printf ("%s\n", result);
```

```
free(result);
free(stack); }
int main() {
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(exp);
    return 0;
}
```

Output

```
abcd^e-fgh*+^*+i-
```

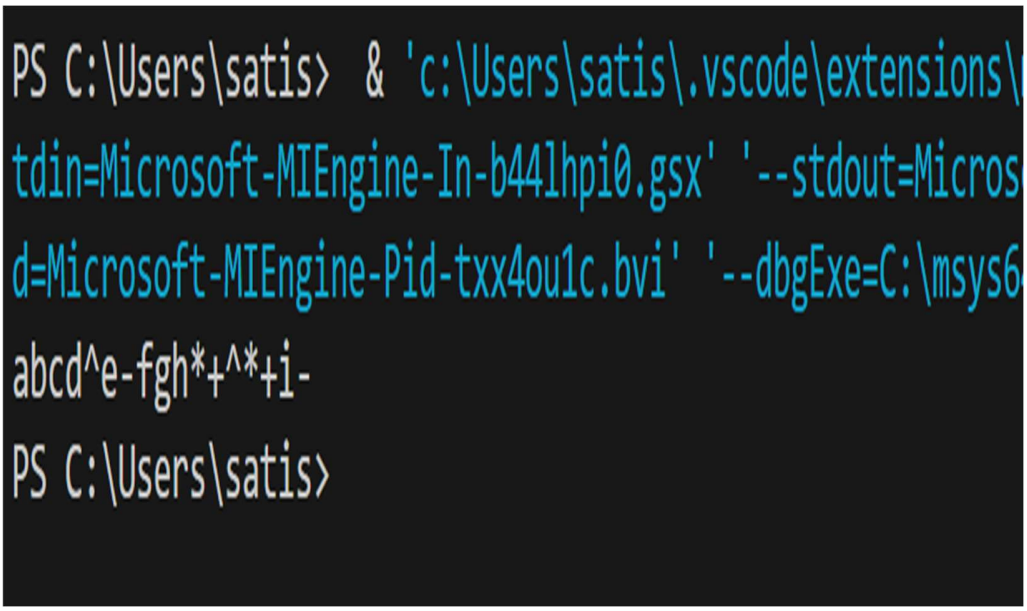
Seen
Gl
28/10/24

Code:

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

int prec(char c){
    if (c == '+' || c == '-') {
        return 1;
    }
    if (c == '*' || c == '/') {
        return 2;
    }
    if (c == '^') {
        return 3;
    }
    return 0;
}

char associativity(char c){
    if(c=='^')
        return 'R';
    return 'L';
}

void infixtopostfix(const char *s){
    int len = strlen(s);
    char result[len+1];
    char stack[len];
    int resultIndex=0;
    int StackIndex=-1;

    if(!result || !stack){
        printf("Memory Allocation Failed \n");
        return;
    }

    for (int i = 0; i < len; i++)
    {
        char c=s[i];
        if ((c>='a' && c<='z') || (c>='A' && c<='Z'))
        {
            result[resultIndex++] = c;
        }
        else if(c=='('){
            stack[++StackIndex]=c;
        }
        else if(c==')'){
            while (StackIndex>=0 && stack[StackIndex]!='('
```

```
            {
                result[resultIndex++]=stack[StackIndex--];
            }
           StackIndex--;
        }
        else{
            while (StackIndex>=0 && (prec(c))<prec(stack[StackIndex]) ||
(prec(c)==prec(stack[StackIndex]) && associativity(c)=='L'))
            {
                result[resultIndex++]=stack[StackIndex--];
            }
            stack[++StackIndex]=c;
        }
    }

    while (StackIndex>=0)
    {
        result[resultIndex++]=stack[StackIndex--];
    }
    result[resultIndex++]='\0';
    printf("%s\n",result);
}

int main(){
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
    infixtopostfix(exp);
    return 0;
}
```

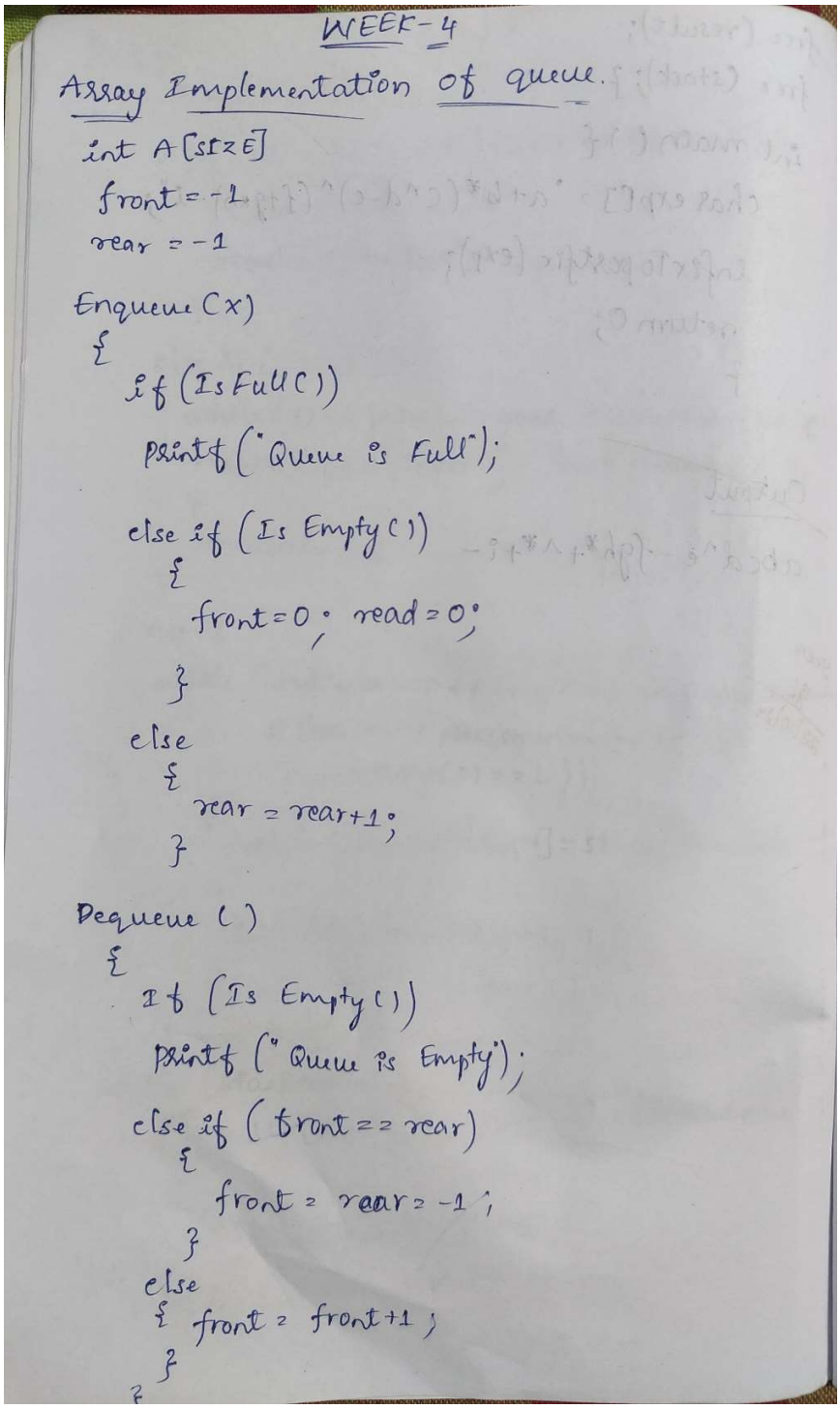
Output:



Program 3

WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display
The program should print appropriate messages for queue empty and queue overflow conditions.

Observation:



WEEK-4

Array Implementation of queue.

```
int A[SIZE]
front = -1
rear = -1

Enqueue(x)
{
  if (IsFull())
   printf("Queue is Full");

  else if (Is Empty())
    {
      front=0; read=0;
    }
  else
    {
      rear = rear+1;
    }

Dequeue()
{
  If (Is Empty())
   printf("Queue is Empty");
  else if (front==rear)
    {
      front = rear = -1;
    }
  else
   { front = front+1;
   }
}
```

```
IsFull()
{
  if (rear == SIZE-1)
    return true;
  else
    return false;
}
IsEmpty()
{
  if (front == -1 && rear == -1)
    return True;
  else
    return false;
}
```

Seen

```
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")
}
```

Output

```
Enqueued 10
Enqueued 20
Enqueued 30
Queue elements : 10 20 30
Dequeued 10
Queue elements : 20 30
Enqueued 40
Enqueued 50
Queue Elements : 20 30 40 50
Dequeued 20
Dequeued 30
Dequeued 40
Dequeued 50
Queue is empty
```

Seen

Seen

14/10/24

Code:

```c
#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)++;

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

```c
}

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:

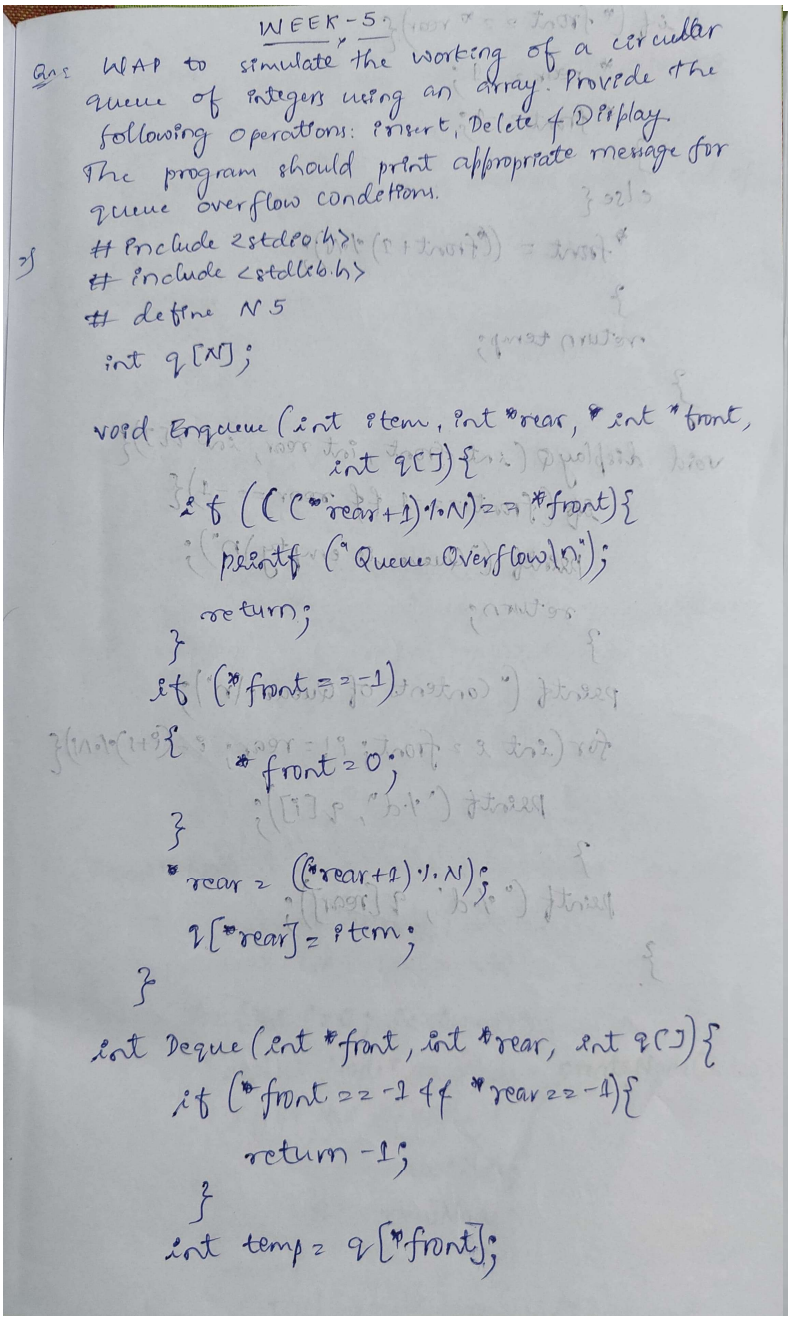
```
C:\Users\satis\practice>gcc Program3.c

C:\Users\satis\practice>a.exe

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 5
5 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 7
7 inserted into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 9
9 inserted into the queue

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

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 5 7 9 10

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

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

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 9 10
```

Program 4

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display.
The program should print appropriate messages for queue empty and queue overflow conditions.

Observation :



WEEK-5

Qn: 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 message for queue overflow condetions.

```
#include <stdio.h>
#include <stdlib.h>
#define N 5
int q[N];

void Enqueue(int item, int *rear, int *front,
             int q[]){
  if (((*rear+1)%N)==*front){
    printf("Queue Overflow\n");
    return;
  }
  if (*front==-1)
    *front=0;
  *rear = ((*rear+1)%N);
  q[*rear]=item;
}

int Deque(int *front, int *rear, int q[]){
  if (*front==-1 || *rear==-1){
    return -1;
  }
  int temp=q[*front];
  if (*front==*rear){
    *rear=-1;
    *front=-1;
  }
  else{
    *front=((*front+1)%N);
  }
  return temp;
}

void display(int front, int rear, int q[]){
  if (front==-1 || rear==-1){
    printf("Queue is empty\n");
    return;
  }
  printf("Contents of Queue:\n");
  for(int i=front; i!=rear; i=(i+1)%N){
    printf("%d", q[i]);
  }
  printf("%d", q[rear]);
}
```

Code:

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

#define N 5

int q[N];

void Enqueue(int item, int *rear,int *front , int q[]) {
    if (((*rear+1)%N) == *front) {
        printf("Queue Overflow \n");
        return;
    }
    if (*front == -1)
    {
        *front = 0;
    }

    *rear = ((*rear + 1)%N);
    q[*rear] = item;
}

int Deque(int *front, int *rear, int q[]) {
    if (*front == -1 && *rear == -1) {
        return -1;
    }
    int temp = q[*front];
    if(*front == *rear){
        *rear = -1;
        *front = -1;
    }
    else{
        *front = ((*front+1)%N);
    }
    return temp;
}

void displayQ(int front, int rear, int q[]) {
    if (front == -1 &&  rear == -1) {
        printf("Queue is empty \n");
        return;
    }
    printf("Contents of Queue:\n");
    for (int i = front; i != rear; i=(i+1)%N) {
        printf("%d ", q[i]);
    }
    printf("%d",q[rear]);
```

```
}

int main() {
    int choice;
    int rear = -1;
    int front = -1;
    int item;
    for (;;) {
        printf("\nMenu:\n1: Enque\n2: Deque\n3: Display\n4: Exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                Enqueue(item, &rear, &front , q);
                break;
            case 2:
                item = Deque(&front, &rear, q);
                if (item == -1)
                    printf("Queue is empty\n");
                else
                    printf("Item deleted = %d \n", item);
                break;
            case 3:
                displayQ(front, rear, q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice, please try again.\n");
        }
    }
    return 0;
}
```

Output:

```
PS C:\Users\satis>  & 'c:\Users\satis\.vscode\extensions\m
tdin=Microsoft-MIEngine-In-jccpmqwv.xwf' '--stdout=Microsc
d=Microsoft-MIEngine-Pid-wxivilue.dcv' '--dbgExe=C:\msys64

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 1

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 4

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 6

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 8

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:3
Contents of Queue:
1 4 6 8
```

```
Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:2
Item deleted = 1

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:2
Item deleted = 4

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:3
Contents of Queue:
6 8
Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 9

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 3
```

```
Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 2

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:1
Enter the item to be inserted: 5
Queue Overflow

Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:3
Contents of Queue:
6 8 9 3 2
Menu:
1: Enque
2: Deque
3: Display
4: Exit
Enter your choice:
```

Program 5

Write a program to Implement Singly Linked List with following operations

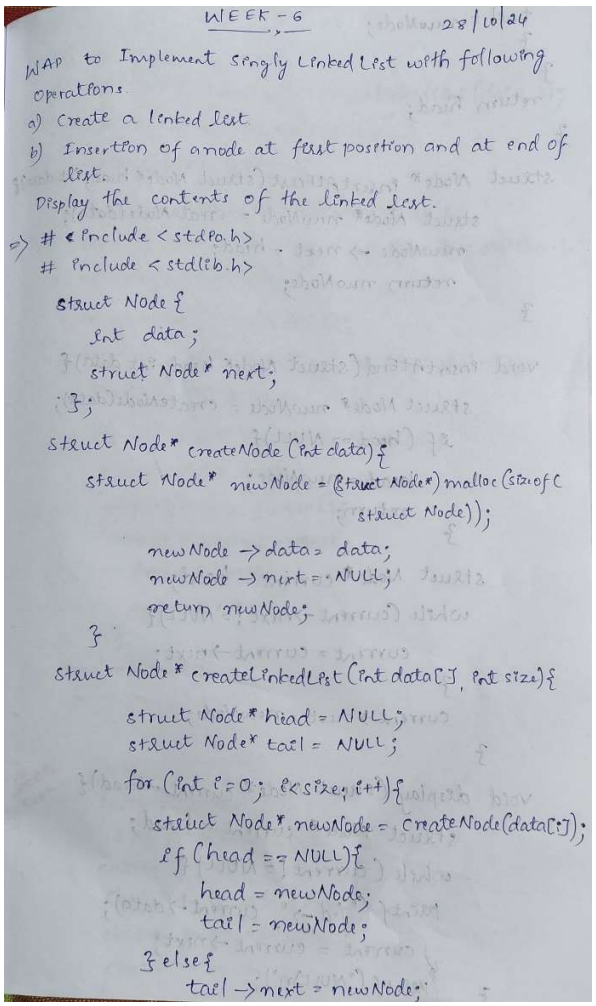
a) Create a linked list.

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

Display the contents of the linked list.

Leetcode problem no.20 (Valid parantheses)

Observation :



WEEK - 6 28/10/24

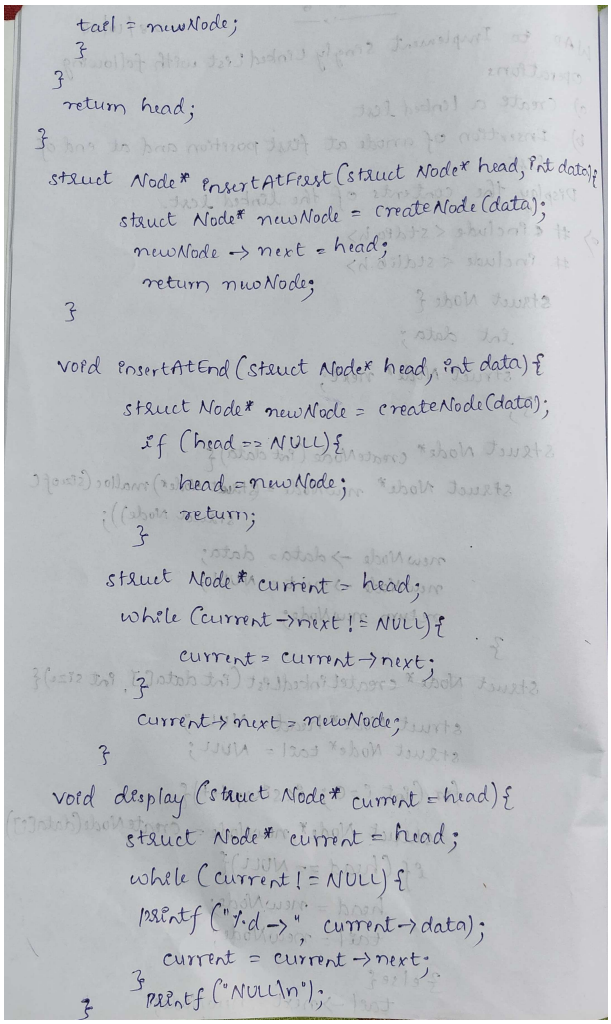
WAP to Implement Singly Linked List with following operations

a) Create a linked list

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

Display the contents of the linked list.

```
#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;
}
struct Node* createLinkedList(int data[], int size){
    struct Node* head = NULL;
    struct Node* tail = NULL;
    for(int i=0; i<size; i++){
        struct Node* newNode = createNode(data[i]);
        if(head == NULL){
            head = newNode;
            tail = newNode;
        }else{
            tail->next = newNode;
```



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

```
int main(){
 int data[] = {1,2,3};
 struct Node* linkedList = createLinkedList(data, 3);
  printf("Initial linked list:\n");
   display(linkedList);
   linkedList = insertAtFirst(linkedList, 0);
   printf("After inserting 0 at the first position:\n");
    display(linkedList);
   insertAtEnd(linkedList, 4);
   printf("After inserting 4 at the end:\n");
   display(linkedList);
    struct Node* current = linkedList;
     struct Node* next;
     while(current != NULL){
       next = current->next;
        free(current);
        current = next;
      }
     return 0;
}
```

Output

Initial linked list:

1 → 2 → 3 → NULL

enter:

choice 1 to addatfirst

choice 2 to addatlast

choice 3 to display

1

enter element to insert at the beginning: 0

enter:

choice 1 to add at fist

choice 2 to addatlast

choice 3 to display

3

0 → 1 → 2 → 3 → NULL

enter:

choice 1 to addatfirst

choice 2 to addatlast

choice 3 to display

2

enter element to insert at the last: 4

enter:

choice 1 to addatfirst

choice 2 to add at last

choice 3 to display

3

0 → 1 → 2 → 3 → 4 → NULL

Code:

```c
 #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;
}

struct Node *createLinkedList(int data[], int size)
{
    struct Node *head = NULL;
    struct Node *tail = NULL;

    for (int i = 0; i < size; i++)
    {
        struct Node *newNode = createNode(data[i]);
        if (head == NULL)
        {
            head = newNode;
            tail = newNode;
        }
        else
        {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return head;
}

struct Node *insertAtFirst(struct Node *head, int data)
{
    struct Node *newNode = createNode(data);
    newNode->next = head;
    return newNode;
}
```

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

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

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

int main()
{

    int data[] = {1, 2, 3};
    struct Node *linkedList = createLinkedList(data, 3);

    printf("Initial linked list:\n");
    display(linkedList);

    int choice;
    printf("Menu:\n1 for addfirst\n2 for addLast\n3 to display\n4 to exit\n");
    printf("Enter your choice");
    scanf("%d", &choice);

    while( choice != 4)
    {

        if (choice == 1)
        {
             int ele;
            printf("enter the ele to add");
           scanf("%d", &ele);
```

```
            linkedList = insertAtFirst(linkedList, ele );

        }
        if (choice == 2)
        {
            int ele;
            printf("enter the ele to add");
           scanf("%d", &ele);

            insertAtEnd(linkedList, ele);

        }

        if (choice == 3)
        {
            display(linkedList);
        }
      printf("Menu:\n1 for addfirst\n2 for addLast\n3 to display\n4 to exit\n");
      printf("Enter your choice");
      scanf("%d", &choice);
    }
}
```

Output:

```
PS C:\Users\satis>  & 'c:\Users\satis\.vscode\extensions\ms-vscode.cppto
tdin=Microsoft-MIEngine-In-5azldy4s.11s' '--stdout=Microsoft-MIEngine-Ou
d=Microsoft-MIEngine-Pid-5d2zx3em.owg' '--dbgExe=C:\msys64\ucrt64\bin\gd
Initial linked list:
1 -> 2 -> 3 -> NULL
Menu:
1 for addfirst
2 for addLast
3 to display
4 to exit
Enter your choice 1
enter the ele to add 0
Menu:
1 for addfirst
2 for addLast
3 to display
4 to exit
Enter your choice 3
0 -> 1 -> 2 -> 3 -> NULL
Menu:
1 for addfirst
2 for addLast
3 to display
4 to exit
Enter your choice 2
enter the ele to add 4
Menu:
1 for addfirst
2 for addLast
3 to display
4 to exit
Enter your choice 3
0 -> 1 -> 2 -> 3 -> 4 -> NULL
Menu:
1 for addfirst
2 for addLast
3 to display
4 to exit
Enter your choice
```

```c
#include <string.h>

bool isValid(char* s) {
    int n = strlen(s);
    char stack[n];
    int top = -1;
    int i = 0;
    while (i < n)
    {
        char c = s[i];
        if (c == '(' || c == '{' || c == '[')
            stack[++top] = c;
        else
        {
            if (top == -1)
                return false;
            char topChar = stack[top--];
            if ((c == ')' && topChar != '(') ||
                (c == '}' && topChar != '{') ||
                (c == ']' && topChar != '['))
                return false;
        }
        i++;
    }
    return top == -1;
}
```

Leet Code Valid Parenthesis

Program 6 :

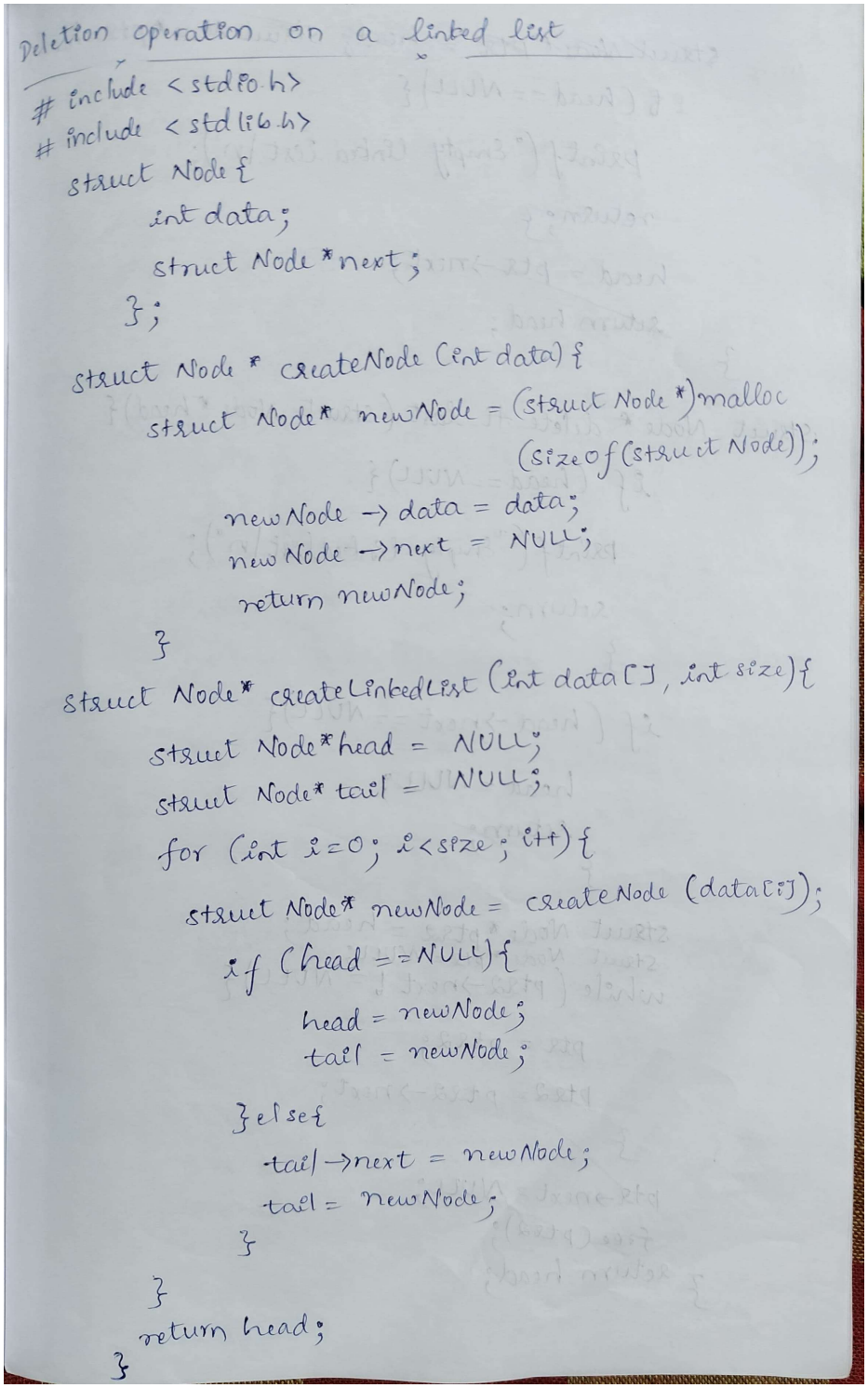
Write a Program to Implement Singly Linked List with following operations

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. Leetcode Problem -- 739 (Daily Temperature)

Observation:



Deletion operation on a linked list

```
# include <stdio.h>
# include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
struct Node * createNode (int data) {
    struct Node* newNode = (struct Node *)malloc
                           (sizeof(struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
struct Node* createLinkedList (int data[], int size){
    struct Node*head = NULL;
    struct Node* tail = NULL;
    for (int i=0; i<size; i++) {
        struct Node* newNode = createNode (data[i]);
        if (head == NULL){
            head = newNode;
            tail = newNode;
        }else{
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}
```

```
struct Node * delete_At_first (struct Node * head) {
    struct Node * ptr = head;
    if (head == NULL) {
        printf ("Empty linked list \n");
        return; }
    head = ptr->next;
    return head;
}
struct Node * delete_At_last (struct Node * head) {
    if (head == NULL) {
        printf ("Empty linked list \n");
        return;
    }
    if (head->next == NULL) {
        head = NULL;
        return;
    }
    struct Node * ptr2 = head;
    struct Node * ptr = NULL;
    while (ptr2->next != NULL) {
        ptr = ptr2;
        ptr2 = ptr2->next;
    }
    ptr->next = NULL;
    free (ptr2);
    return head;
}
```

```
struct Node * deletespecified (struct Node * head, int value);
    if (head == NULL) {
        printf ("List is empty\n");
        return;
    }
    if (head->data == value) {
        struct Node * temp = head;
        head = head->next;
        free (temp);
        return;
    }
    struct Node * temp = head;
    struct Node * prev = NULL;
    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf ("Element %d not found.\n", value);
        return;
    }
    prev->next = temp->next;
    free (temp);
    return head;
}
```

Seen

o/p seen

Code:

```c
 #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;
}

struct Node *createLinkedList(int data[], int size)
{
    struct Node *head = NULL;
    struct Node *tail = NULL;

    for (int i = 0; i < size; i++)
    {
        struct Node *newNode = createNode(data[i]);
        if (head == NULL)
        {
            head = newNode;
            tail = newNode;
        }
        else
        {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return head;
}

void display(struct Node *head)
{
    struct Node *current = head;
    while (current != NULL)
    {
        printf("%d -> ", current->data);
        current = current->next;
```

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

struct Node *delete_at_first(struct Node *head)
{
    if (head == NULL)
    {
        printf("Empty linked list\n");
        return head;
    }

    struct Node *ptr = head;
    head = ptr->next;
    free(ptr);
    return head;
}

struct Node *delete_at_last(struct Node *head)
{
    if (head == NULL)
    {
        printf("Empty linked list\n");
        return head;
    }

    if (head->next == NULL)
    {
        free(head);
        return NULL;
    }

    struct Node *ptr2 = head;
    struct Node *ptr = NULL;
    while (ptr2->next != NULL)
    {
        ptr = ptr2;
        ptr2 = ptr2->next;
    }
    ptr->next = NULL;
    free(ptr2);
    return head;
}

struct Node *deletespecified(struct Node *head, int value)
{
    if (head == NULL)
    {
        printf("List is empty!\n");
        return head;
```

```
    }

    if (head->data == value)
    {
        struct Node *temp = head;
        head = temp->next;
        free(temp);
        return head;
    }

    struct Node *current = head;
    while (current->next != NULL && current->next->data != value)
    {
        current = current->next;
    }

    if (current->next == NULL)
    {
        printf("Value %d not found in the list!\n", value);
    }
    else
    {
        struct Node *temp = current->next;
        current->next = current->next->next;
        free(temp);
    }
    return head;
}

int main()
{
    int data[] = {1, 2, 3, 4, 5};
    struct Node *linkedList = createLinkedList(data, 5);

    printf("Initial linked list:\n");
    display(linkedList);

    int choice;
    printf("Menu\n1. Delete first node\n2. Delete last node\n3. Delete specified
node\n4. Display list\n5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    while (choice != 5)
    {
        if (choice == 1)
        {
            linkedList = delete_at_first(linkedList);
            printf("After deleting the first node:\n");
            display(linkedList);
```

```
        }
        if (choice == 2)
        {
            linkedList = delete_at_last(linkedList);
            printf("After deleting the last node:\n");
            display(linkedList);
        }
        if (choice == 3)
        {
            int value;
            printf("Enter the value to be deleted: ");
            scanf("%d", &value);
            linkedList = deletespecified(linkedList, value);
            printf("After deleting the specified node:\n");
            display(linkedList);
        }
        if (choice == 4)
        {
            display(linkedList);
        }
        if (choice == 5)
        {
            exit(0);
        }

        printf("Menu\n1. Delete first node\n2. Delete last node\n3. Delete specified
node\n4. Display list\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
    }

    return 0;
}
```

Output:

```
C:\Users\satis\practice>gcc Program5.c

C:\Users\satis\practice>a.exe
Initial linked list:
1 -> 2 -> 3 -> 4 -> 5 -> NULL
Menu
1. Delete first node
2. Delete last node
3. Delete specified node
4. Display list
5. Exit
Enter your choice: 1
After deleting the first node:
2 -> 3 -> 4 -> 5 -> NULL
Menu
1. Delete first node
2. Delete last node
3. Delete specified node
4. Display list
5. Exit
Enter your choice: 3
Enter the value to be deleted: 3
After deleting the specified node:
2 -> 4 -> 5 -> NULL
Menu
1. Delete first node
2. Delete last node
3. Delete specified node
4. Display list
5. Exit
Enter your choice: 2
After deleting the last node:
2 -> 4 -> NULL
Menu
1. Delete first node
2. Delete last node
3. Delete specified node
4. Display list
5. Exit
Enter your choice: 4
2 -> 4 -> NULL
```

```c
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
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 !=-1 && temperatures[i] > temperatures[stack[top]]) {
            int j = stack[top--];
            answer[j] = i - j;
        }
        stack[++top] = i;

    free(stack);
    return answer;

}
```

Leet Code Daily Temperatures

Program 7

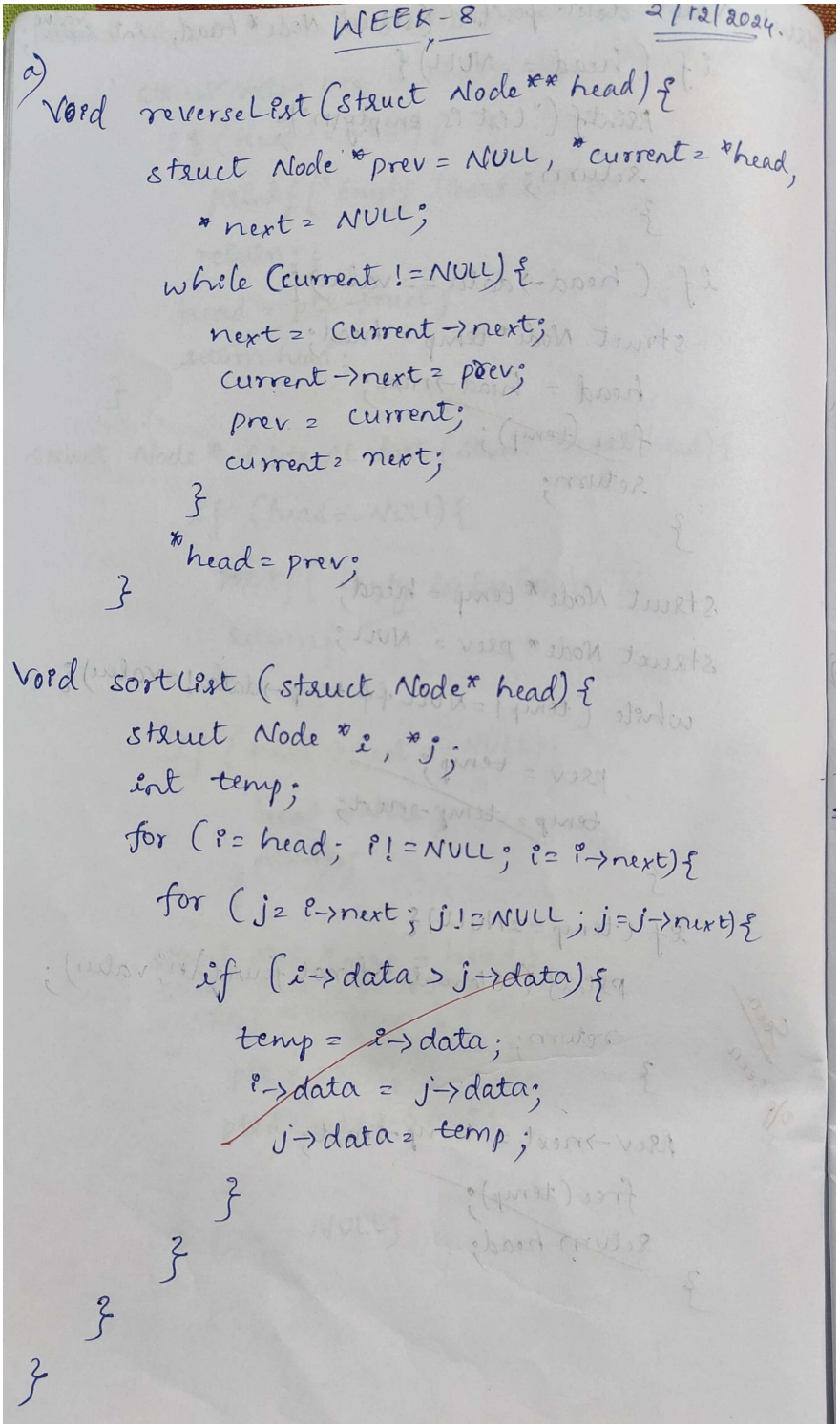
a) Write a Program to Implement Single Link List with following operations:

(i)Sort the linked list,

(ii) Reverse the linked list,

(iii)Concatenation of two linked lists.

Observation:



WEEK-8 21/12/2024.

a)

```
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 sortList (struct Node* head){
    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 concatenateList (struct Node **head1, struct Node **head2){
    if (*head1 == NULL){
        *head1 = head2;
    } else {
        struct Node *temp = *head1;
        while (temp->next != NULL){
            temp = temp->next;
        }
        temp->next = head2;
    }
}
```

b)Write a Program to Implement Single Link List to simulate
(i)Stack
(ii)Queue Operations.

Observation:

```
b) WAP to implement Single Link List to stimulate
   stack & Queue Operations.
=> struct Node *top = NULL;
void push (int x) {
    struct Node *newNode = (struct Node*) malloc (sizeof
                           (struct Node));
    newNode->data = x;
    newNode->next = top;
    top = newNode;
}
void pop () {
    struct Node *temp;
    temp = top;
    if (top == NULL) {
        printf ("stack Underflow\n");
    }
```

```
    else {
        printf("Popped element is %d\n", top->data);
        top = top->next;
        free(temp);
    }
}
```

- Queue Operations:

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

void enqueue(int x) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct
    newNode->data = x;
    newNode->next = NULL;
    if (front == NULL && rear == NULL) {
        front = rear = newNode;
    }
    else {
        rear->next = newNode;
        rear = newNode;
    }
}
```

```
void dequeue() {
    struct Node *temp;
     temp = front;
    if (front == NULL && rear == NULL) {
        printf("Queue is empty\n");
    } else if (front == rear) {
        front = rear = NULL;
    } else {
        front = front->next;
        free(temp);
    }
}
```

23/12/24

Code:

```c
#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 insertEnd(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;
    }
}


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


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

```
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}


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


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


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


    insertEnd(&list1, 5);
    insertEnd(&list1, 1);
    insertEnd(&list1, 9);
    insertEnd(&list1, 3);


    insertEnd(&list2, 8);
    insertEnd(&list2, 2);
    insertEnd(&list2, 4);


    printf("List 1: ");
    printList(list1);
```

```
    printf("List 2: ");
    printList(list2);

    sortList(list1);
    printf("\nList 1 after sorting: ");
    printList(list1);


    reverseList(&list2);
    printf("\nList 2 after reversing: ");
    printList(list2);


    concatenateLists(&list1, list2);
    printf("\nList 1 after concatenation with List 2: ");
    printList(list1);

    return 0;
}
```

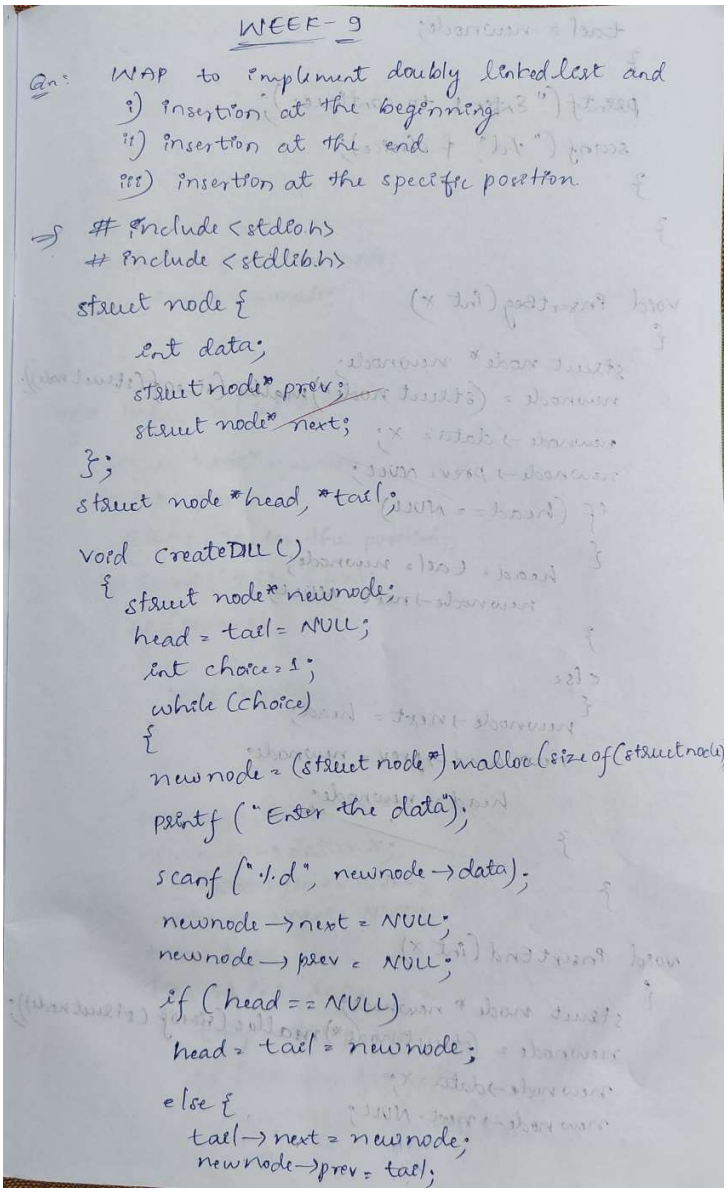
Output:

```
PS C:\Users\satis\OneDrive\Desktop>  & 'c:\Users\satis\.vscode\extensions\ms-vscode.cpptools-1.22.11-win32-x64\debugAdapters\bin\W
indowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-rktevcfs.pxp' '--stdout=Microsoft-MIEngine-Out-fqep0jvm.znr' '--stderr=Mic
rosoft-MIEngine-Error-u5zpearj.1nu' '--pid=Microsoft-MIEngine-Pid-l15zr4xd.ahj' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpr
eter=mi'
List 1: 5 -> 1 -> 9 -> 3 -> NULL
List 2: 8 -> 2 -> 4 -> NULL

List 1 after sorting: 1 -> 3 -> 5 -> 9 -> NULL

List 2 after reversing: 4 -> 2 -> 8 -> NULL

List 1 after concatenation with List 2: 1 -> 3 -> 5 -> 9 -> 4 -> 2 -> 8 -> NULL
```

Program 8

Write a Program 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

Observation:



WEEK-9

Q: WAP to implement doubly linked list and
i) insertion at the beginning
ii) insertion at the end
iii) insertion at the specific position

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node* prev;
    struct node* next;
};
struct node *head, *tail;
void createDLL()
{
    struct node* newnode;
    head = tail = NULL;
    int choice = 1;
    while (choice)
    {
    newnode = (struct node*) malloc (sizeof(struct node));
    printf ("Enter the data");
    scanf ("%d", newnode->data);
    newnode->next = NULL;
    newnode->prev = NULL;
    if (head == NULL)
     head = tail = newnode;
    else {
      tail->next = newnode;
      newnode->prev = tail;
      tail = newnode;
    }
    printf ("Enter 1 to continue:");
    scanf ("%d", &choice);
    }
}

void InsertBeg (int x)
{
    struct node* newnode;
    newnode = (struct node*) malloc (sizeof(struct
    newnode->data = x;
    newnode->prev = NULL;
    if (head == NULL)
    {
      head = tail = newnode;
      newnode->next = NULL;
    }
    else
    {
      newnode->next = head;
      head->prev = newnode;
      head = newnode;
    }
}

void InsertEnd (int x)
{
    struct node* newnode;
    newnode = (struct node*) malloc (sizeof(struct
    newnode->data = x;
    newnode->next = NULL;
```

```
if (head == NULL)
{
    head = tail = newnode;
    newnode->prev = NULL;
}
else
{ newnode->prev = tail;
  tail->next = newnode;
    tail = newnode;
}
}
void insertPos (int x)
{
  struct node* newnode, *temp;
  int pos;
  printf ("Enter the position");
  scanf ("%d", &pos);
  if (pos==1)
  {
    void insertBeg(x);
  }
  else
  {
  newnode = (struct node*) malloc (sizeof(struct node));
  newnode -> data = x;
  newnode -> prev = NULL;
  newnode -> next = NULL;
  temp = head;
  for (int i=1, i<pos-1; i++)
  { temp = temp->next;
    if (temp == tail->next)
    { printf("There are less than %d nodes", pos);
      return;
  } }
```

```
newnode->prev = temp;
newnode->next = temp->next;
temp->next = newnode;
newnode->next->prev = newnode;
}
}
```

Seen

Output:

```
Enter the data: 1
enter 1 to continue: 1
Enter the data: 2
enter 1 to continue: 1
Enter the data: 3
enter 1 to continue: 2
Menu:
1. Insert at Beginning
2. Insert at specific position
3. Insert at End
4. Display list
5. Exit
Enter your choice: 1
Enter value to insert at the beginning: 4
Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display list
5. Exit.
Enter your choice: 4
4 <-> 1 <-> 2 <-> 3 <-> NULL
```

```
Menu:
1. Insert at Beginning
2. Insert at specific Position
3. Insert at End
4. Display list
5. Exit
Enter your choice: 2
Enter value to insert: 5
Enter the position: 2
Menu:
1. Insert at Beginning
2. Insert at specific position
3. Insert at End
4. Display list
5. Exit
Enter your choice: 4
4 <-> 5 <-> 1 <-> 2 <-> 3 <-> NULL

Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display list
5. Exit
Enter your choice: 3
Enter value to insert at the end: 6
Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display list
5. Exit.
Enter your choice: 4
4 <-> 5 <-> 1 <-> 2 <-> 3 <-> 6 <-> NULL
```

Seen

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head, *tail;
void createDLL(){
    struct node *newnode;
    head = tail = NULL;
    int choice = 1;
    while(choice==1){
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter the data:");
        scanf("%d", &newnode->data);
        newnode->next = NULL;
        newnode->prev = NULL;
        if(head == NULL){
        head = tail = newnode;
        }
        else{
        tail->next = newnode;
        newnode->prev = tail;
        tail = newnode;
        }
        printf("enter 1 to continue:");
        scanf("%d", &choice);
    }
}
void insertBeg(int x){
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = x;
    newnode->prev = NULL;
    if(head == NULL){
    head = tail = newnode;
    newnode->next = NULL;
    }
    else{
    newnode->next = head;
    head->prev = newnode;
    head = newnode;
    }
}
void insertEnd(int x){
    struct node *newnode;
    newnode = (struct node *)malloc(sizeof(struct node));
```

```
    newnode->data = x;
    newnode->next = NULL;
    if(head == NULL){
    head = tail = newnode;
    newnode->prev = NULL;
    }
    else{
    newnode->prev = tail;
    tail->next = newnode;
    tail = newnode;
    }
}
void insertPos(int x){
    struct node *newnode, *temp;
    int pos;
    printf( "Enter the position:");
    scanf("%d", &pos);
    if(pos == 1){
        insertBeg(x);
    }
    else{
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->data = x;
        newnode->next = NULL;
        newnode->prev = NULL;
        temp = head;
        for(int i = 1; i < pos-1 ; i++){
            temp=temp->next;
            if(temp == tail->next){
                printf("There are less than %d nodes\n",pos);
                return;
            }
        }
        newnode->prev = temp;
        newnode->next = temp->next;
        temp->next = newnode;
        newnode->next->prev = newnode;
    }
}
void display(){
    struct node *temp;
    if(head == NULL){
        printf("list is empty\n");
    }
    else{
        temp = head;
        while(temp != NULL){
            printf("%d<->", temp->data);
            temp = temp->next;
        }
```

```
        printf("NULL\n");
    }
}
int main() {
 createDLL();
 int choice, value, position;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at Specific Position\n");
        printf("3. Insert at End\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertPos(value);
                break;

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

            case 4:
                display();
                break;

            case 5:
                printf("Exiting...\n");
                return 0;

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

Output:

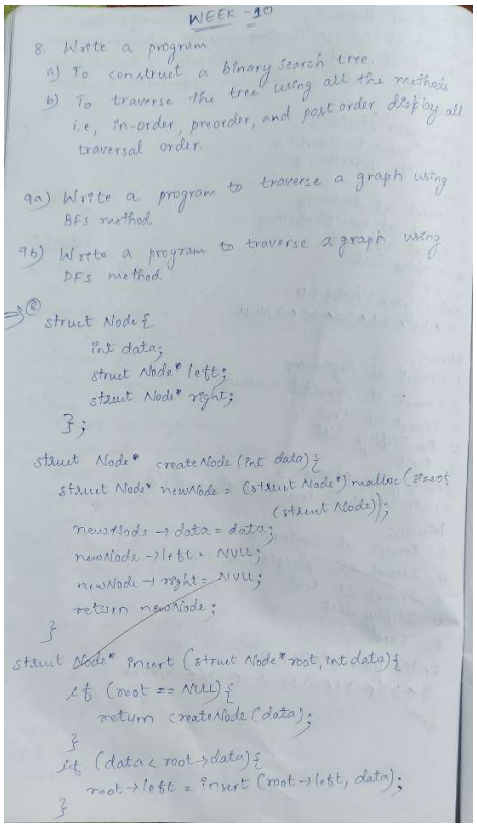
```
Enter the data:1
enter 1 to continue:1
Enter the data:2
enter 1 to continue:1
Enter the data:3
enter 1 to continue:2

Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 1
Enter value to insert at the beginning: 4

Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 4
4<->1<->2<->3<->NULL

Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 2
Enter value to insert: 5
Enter the position:2

Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 4
4<->5<->1<->2<->3<->NULL

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

```
Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 3
Enter value to insert at the end: 6

Menu:
1. Insert at Beginning
2. Insert at Specific Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 4
4<->5<->1<->2<->3<->6<->NULL
```

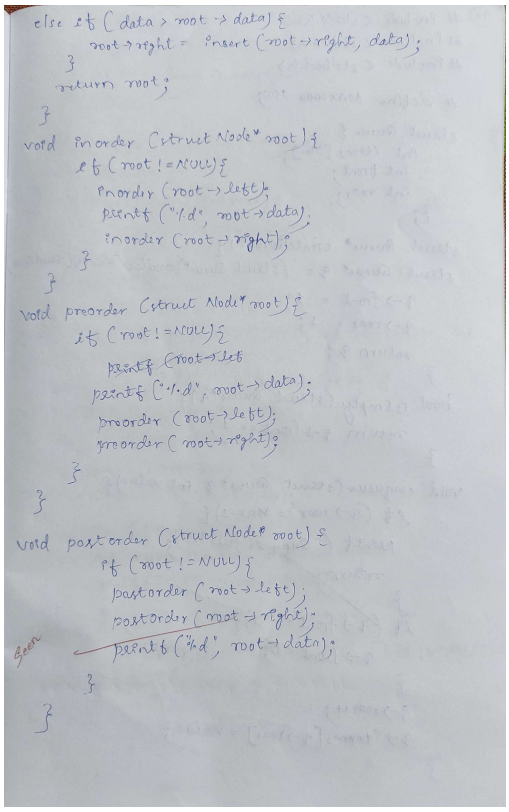
Program 9

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

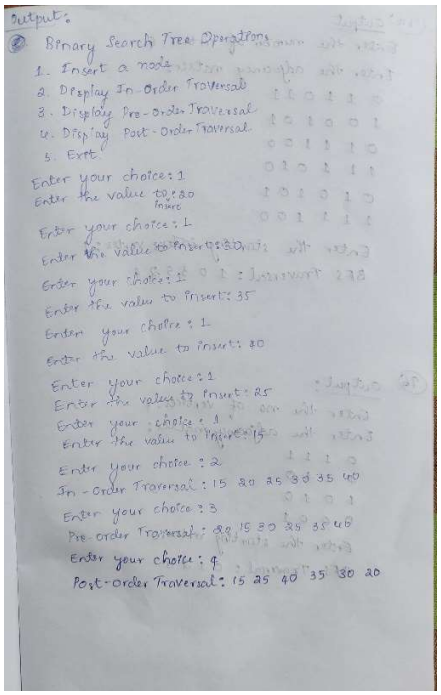
Observation:



```
WEEK - 10
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 display all
   traversal order.

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

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

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

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

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



```
    else if (data > root->data){
        root->right = insert (root->right, data);
    }
    return root;
}

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

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

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



```
Output:

Binary Search Tree Operations
1. Insert a node
2. Display In-Order Traversal
3. Display Pre-Order Traversal
4. Display Post-Order Traversal
5. Exit

Enter your choice: 1
Enter the value to insert: 20
Enter your choice: 1
Enter the value to insert: 30
Enter your choice: 1
Enter the value to insert: 35
Enter your choice: 1
Enter the value to insert: 40
Enter your choice: 1
Enter the value to insert: 25
Enter your choice: 1
Enter the value to insert: 15
Enter your choice: 2
In-Order Traversal: 15 20 25 30 35 40
Enter your choice: 3
Pre-order Traversal: 20 15 30 25 35 40
Enter your choice: 4
Post-Order Traversal: 15 25 40 35 30 20
```

Code:

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


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

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


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


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


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

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


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

    printf("Binary Search Tree Operations:\n");
    printf("1. Insert a node\n");
    printf("2. Display In-order Traversal\n");
    printf("3. Display Pre-order Traversal\n");
    printf("4. Display Post-order Traversal\n");
    printf("5. Exit\n");

    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Post-order Traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                printf("Exiting...\n");
                exit(0);
            default:
```

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

    return 0;
}
```

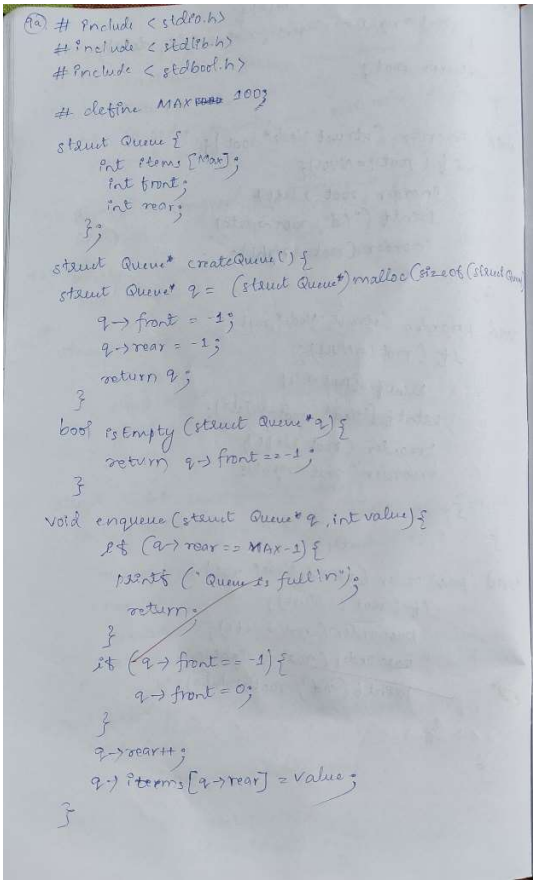
Output:

```
Binary Search Tree Operations:
1. Insert a node
2. Display In-order Traversal
3. Display Pre-order Traversal
4. Display Post-order Traversal
5. Exit

Enter your choice: 1
Enter the value to insert: 20

Enter your choice: 1
Enter the value to insert: 30

Enter your choice: 1
Enter the value to insert: 35

Enter your choice: 1
Enter the value to insert: 40

Enter your choice: 1
Enter the value to insert: 25

Enter your choice: 1
Enter the value to insert: 15

Enter your choice: 2
In-order Traversal: 15 20 25 30 35 40

Enter your choice: 3
Pre-order Traversal: 20 15 30 25 35 40

Enter your choice: 4
Post-order Traversal: 15 25 40 35 30 20
```

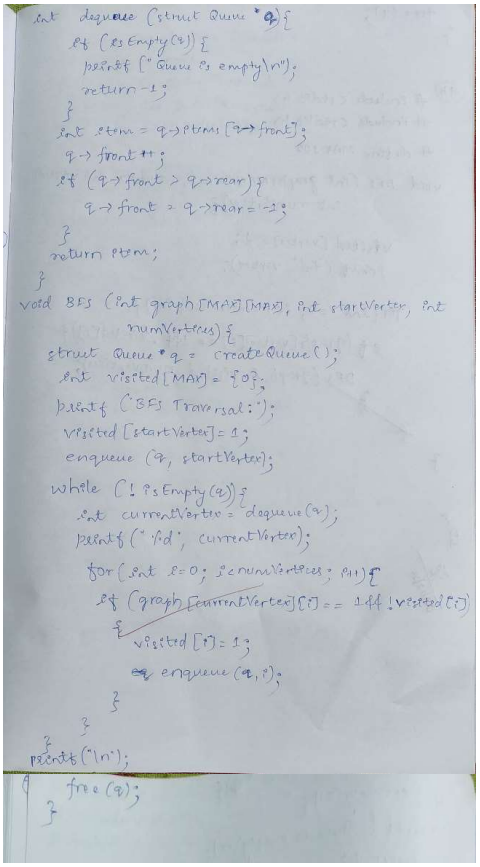
Program 10:

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

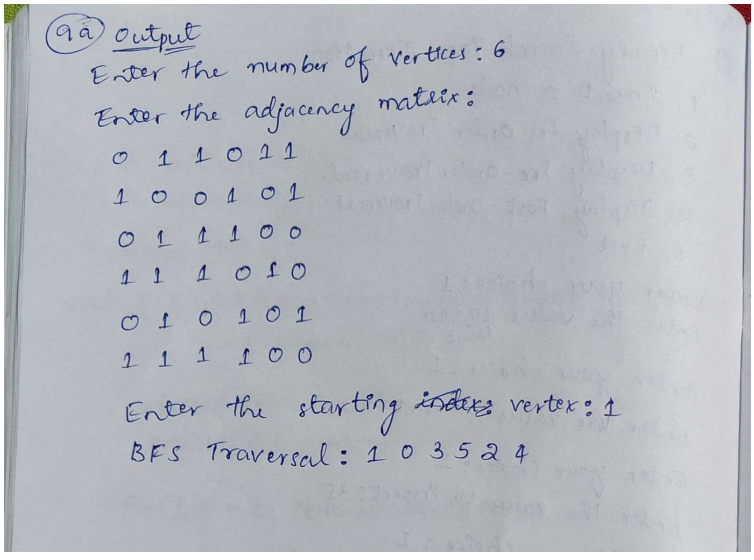
Observation:



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

#define MAX 100

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

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

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

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX-1) {
        printf("Queue is full\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 is empty\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][MAX], int startVertex, int numVertices) {
    struct Queue* q = createQueue();
    int visited[MAX] = {0};
    printf("BFS Traversal: ");
    visited[startVertex] = 1;
    enqueue(q, startVertex);
    while (!isEmpty(q)) {
        int currentVertex = dequeue(q);
        printf("%d ", currentVertex);
        for (int i=0; i<numVertices; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(q, i);
            }
        }
    }
    printf("\n");
    free(q);
}
```



9a) Output

```
Enter the number of vertices: 6
Enter the adjacency matrix:
0 1 1 0 1 1
1 0 0 1 0 1
0 1 1 1 0 0
1 1 1 0 1 0
0 1 0 1 0 1
1 1 1 1 0 0
Enter the starting vertex: 1
BFS Traversal: 1 0 3 5 2 4
```

Code:

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

#define MAX 100
struct Queue {
    int items[MAX];
    int front;
    int rear;
};

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

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

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue is full\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 is empty\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][MAX], int startVertex, int numVertices) {
    struct Queue* q = createQueue();
```

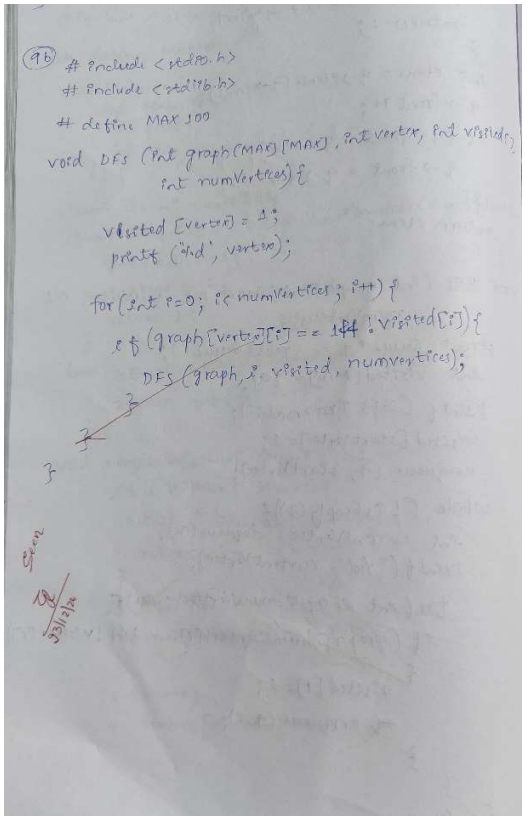
```c
    int visited[MAX] = {0};

    printf("BFS Traversal: ");
    visited[startVertex] = 1;
    enqueue(q, startVertex);

    while (!isEmpty(q)) {
        int currentVertex = dequeue(q);
        printf("%d ", currentVertex);

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

    free(q);
}

int main() {
    int graph[MAX][MAX], numVertices, startVertex;

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

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

    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    BFS(graph, startVertex, numVertices);

    return 0;
}
```

Output:

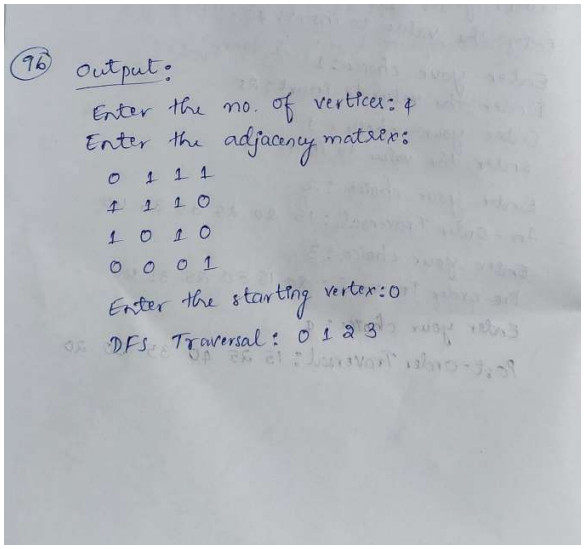
```
Enter the number of vertices: 6
Enter the adjacency matrix:
0 1 1 0 1 1
1 0 0 1 0 1
0 1 1 1 0 0
1 1 1 0 1 0
0 1 0 1 0 1
1 1 1 1 0 0
Enter the starting vertex: 1
BFS Traversal: 1 0 3 5 2 4
```

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

Observation:



```
(9b) #include <stdio.h>
     #include <stdlib.h>
     #define MAX 100
     void DFS (int graph[MAX][MAX], int vertex, int visited[],
               int numVertices){
         visited[vertex] = 1;
         printf("%d", vertex);
         for(int i=0; i<numVertices; i++){
             if(graph[vertex][i] == 1 && !visited[i]){
                 DFS(graph, i, visited, numVertices);
             }
         }
     }
```



```
(9b) Output:
Enter the no. of vertices: 4
Enter the adjacency matrix:
0 1 1 1
1 1 1 0
1 0 1 0
0 0 0 1
Enter the starting vertex: 0
DFS Traversal: 0 1 2 3
```

Code:

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

#define MAX 100


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

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


int main() {
    int graph[MAX][MAX], numVertices, startVertex;
    int visited[MAX] = {0};

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

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

    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    printf("DFS Traversal: ");
    DFS(graph, startVertex, visited, numVertices);
    printf("\n");

    return 0;
```

Output:

```
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 1 1
1 1 1 0
1 0 1 0
0 0 0 1
Enter the starting vertex: 0
DFS Traversal: 0 1 2 3
```