VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

Spandana M R (1BM23CS337)

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 **Spandana M R(1BM23CS337),** 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.

Prasad G R Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

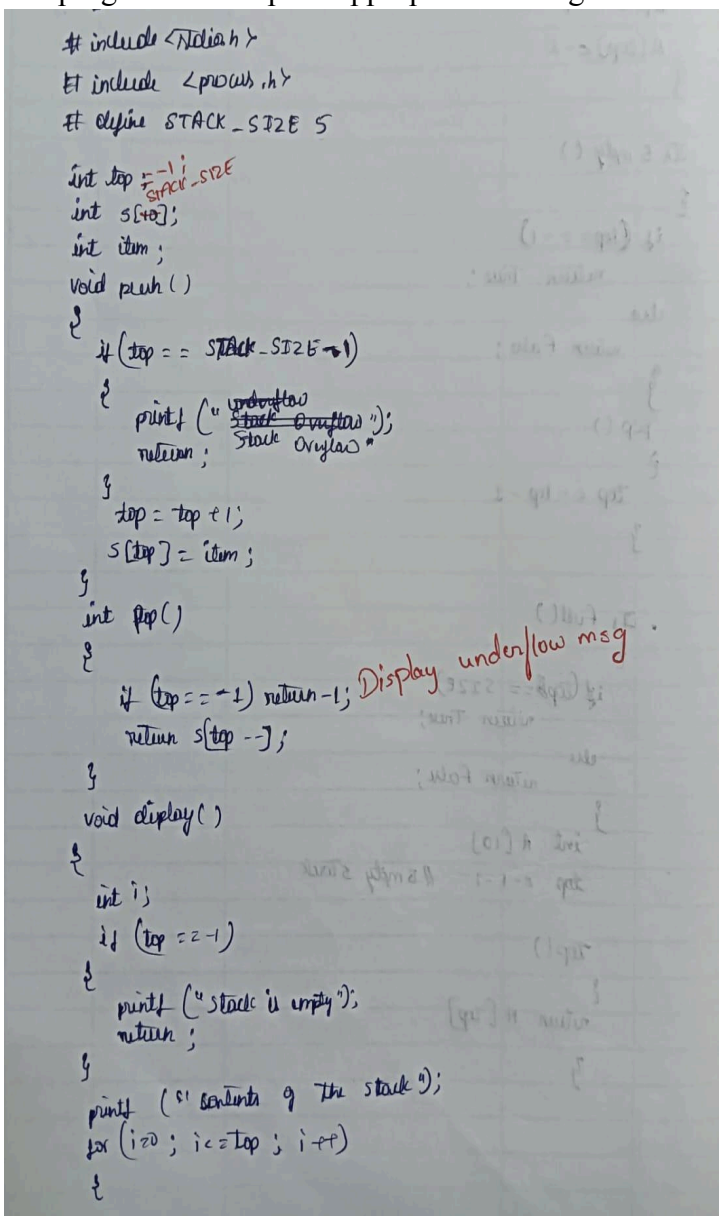
Index

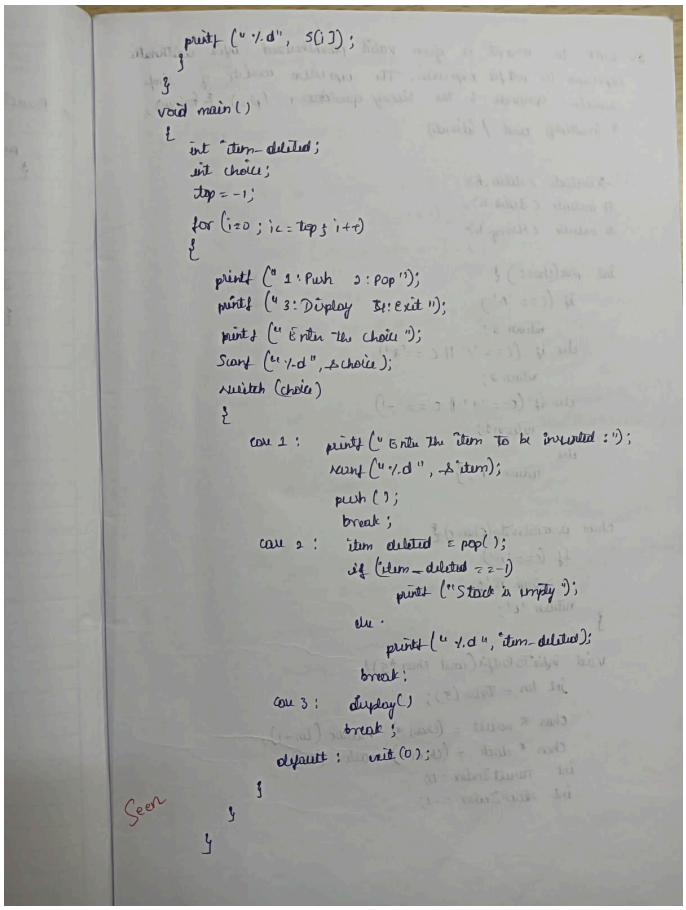
Github Link:
https://github.com/Spandana-mr/DS.C

Lab Programs:

1.Write a program to simulate the working of stack using an

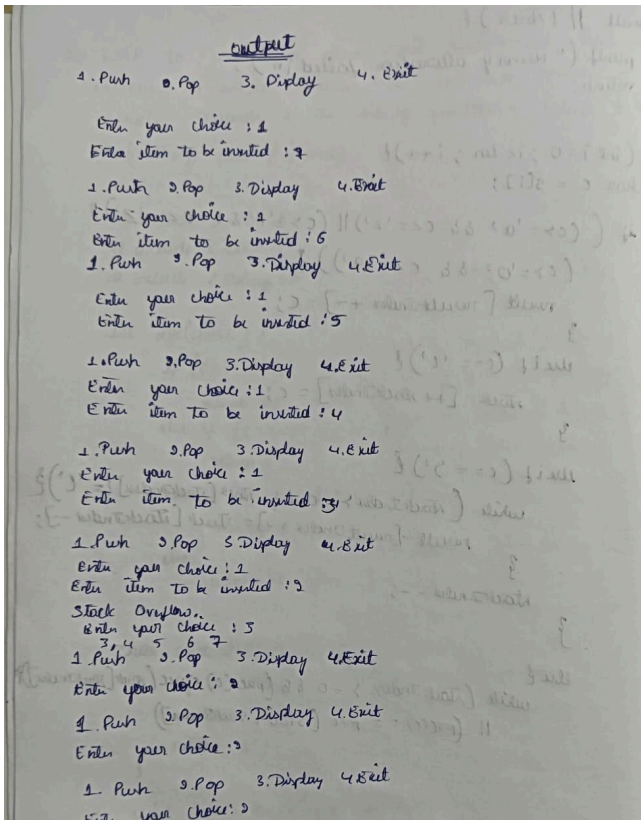
array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow





```
        printf ("%d", S[i]);
    }
}
void main()
{
    int item_deleted;
    int choice;
    top = -1;
    for (i=0; i<=top; i++)
    {
        printf ("1:Push 2:Pop");
        printf ("3:Display 4:Exit");
        printf ("Enter the choice");
        scanf ("%d", &choice);
        switch (choice)
        {
            case 1: printf ("Enter the item to be inserted:");
                    scanf ("%d", &item);
                    push();
                    break;
            case 2: item_deleted = pop();
                    if (item_deleted == -1)
                        printf ("Stack is empty");
                    else
                        printf ("%d", item_deleted);
                    break;
            case 3: display()
                    break;
            default: exit(0);
        }
    }
}
```

Seen



output

1. Push 2. Pop 3. Display 4. Exit

Enter your choice : 1
Enter item to be inserted : 7

1. Push 2. Pop 3. Display 4. Exit
Enter your choice : 1
Enter item to be inserted : 6
1. Push 2. Pop 3. Display 4. Exit

Enter your choice : 1
Enter item to be inserted : 5

1. Push 2. Pop 3. Display 4. Exit
Enter your choice : 1
Enter item to be inserted : 4

1. Push 2. Pop 3. Display 4. Exit
Enter your choice : 1
Enter item to be inserted : 3

1. Push 2. Pop 3. Display 4. Exit
Enter your choice : 1
Enter item to be inserted : 2
Stack Overflow..
Enter your choice : 3
3, 4 5 6 7
1. Push 2. Pop 3. Display 4. Exit
Enter your choice : 2
1. Push 2. Pop 3. Display 4. Exit
Enter your choice : 2

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

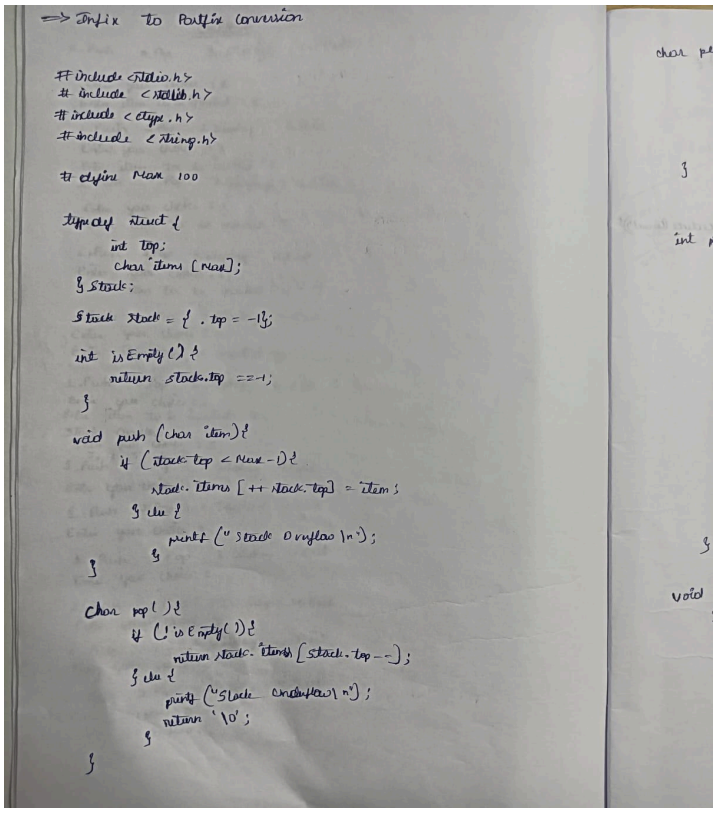
```c
#include<stdio.h>
#include<stdlib.h>
#define size 3

int top=-1;
int stack[size];
int item;

void push(){
    if(top==size-1){
        printf("Stack Overload\n");
    }
    else{
        top+=1;
        stack[top]=item;
    }
}

int pop(){
    if(top==-1){
        printf("Stack Underflow\n");
    }
    else{
        return stack[top--];
    }
}

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

void main(){
    int choice;
```

```
    while(1){
        printf("Enter your options:\n");
        printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1:printf("Enter the element to be pushed
in:");scanf("%d",&item);push();break;
            case 2:if(top==-1){
                printf("stack is empty!\n");
            }else{
                printf("%d popped from stack\n", stack[top]);
            }
            pop();
            break;
            case 3:display();
            break;
            case 4:exit(0);
        }
    }
}
```

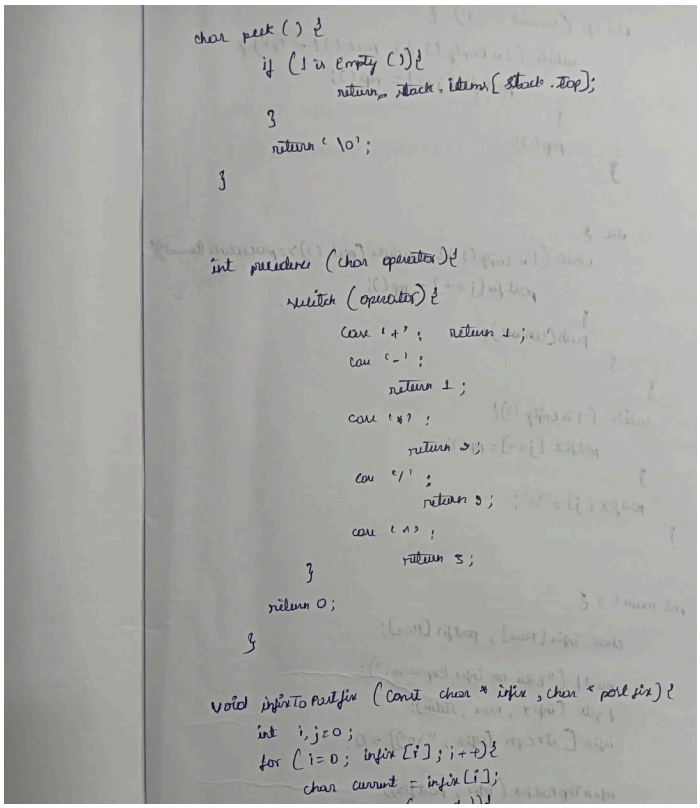
```
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the element to be pushed in:1
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the element to be pushed in:2
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the element to be pushed in:3
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the element to be pushed in:4
Stack Overload
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:3
Content of the stacks:1 2 3
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
```

```
Enter your choice:2
3 popped from stack
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:2
2 popped from stack
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:2
1 popped from stack
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:2
stack is empty!
Stack Underflow
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:3
Stack is empty!Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:4
PS C:\Users\Spandana\OneDrive\Documents\dsa programs\output>
```

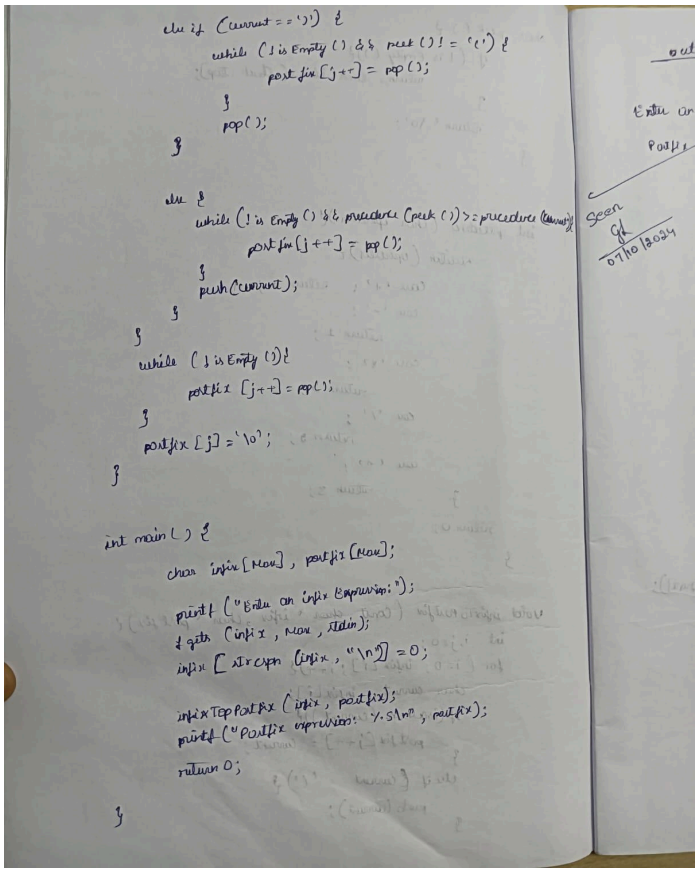
2.WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus),- (minus), * (multiply) and / (divide)



```
==> Infix to Postfix conversion

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

#define Max 100

typedef struct {
    int top;
    char items[Max];
} Stack;

Stack stack = { .top = -1 };

int isEmpty() {
    return stack.top == -1;
}

void push(char item) {
    if (stack.top < Max - 1) {
        stack.items[++stack.top] = item;
    } else {
        printf("Stack Overflow\n");
    }
}

char pop() {
    if (!isEmpty()) {
        return stack.items[stack.top--];
    } else {
        printf("Stack Underflow\n");
        return '\0';
    }
}
```



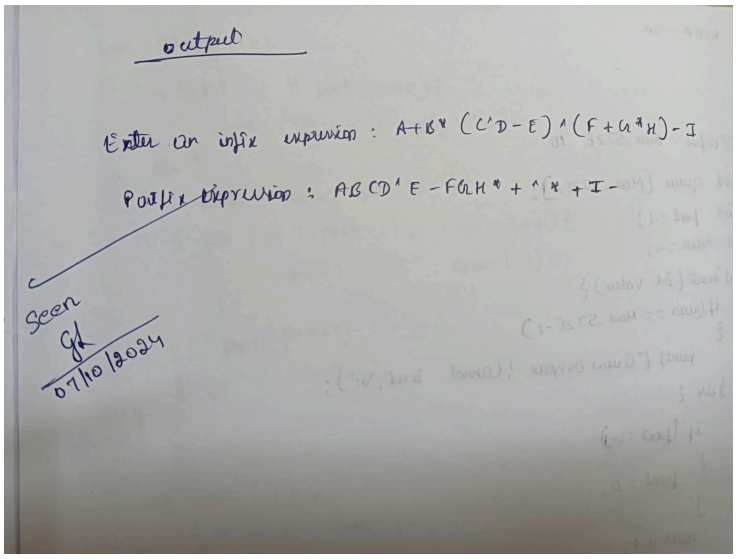
```
char peek() {
    if (!isEmpty()) {
        return stack.items[stack.top];
    }
    return '\0';
}

int precedence(char operator) {
    switch (operator) {
        case '+': return 1;
        case '-':
            return 1;
        case '*':
            return 2;
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return 0;
}

void infixToPostfix(const char *infix, char *postfix) {
    int i, j=0;
    for (i=0; infix[i]; i++) {
        char current = infix[i];
```



```
    else if (current == ')') {
        while (!isEmpty() && peek() != '(') {
            postfix[j++] = pop();
        }
        pop();
    }

    else {
        while (!isEmpty() && precedence(peek()) >= precedence(current)) {
            postfix[j++] = pop();
        }
        push(current);
    }
}
while (!isEmpty()) {
    postfix[j++] = pop();
}
postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];

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

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

    return 0;
}
```

Enter an
Postfix

Seen
07/10/2024



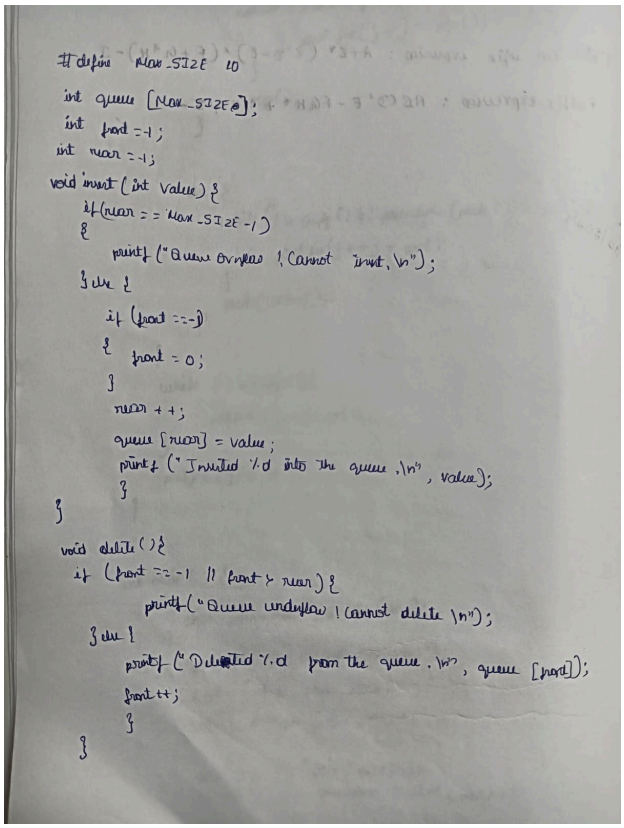

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define MAX 5

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

void push(char c) {
    if (top < MAX - 1) {
        stack[++top] = c;
    }
}


char pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return '\0';
}


char peek() {
    if (top >= 0) {
        return stack[top];
```

```
    }
    return '\0';
}


int precedence(char c) {
    switch (c) {
        case '+': return 1;
        case '-': return 1;
        case '*': return 2;
        case '/': return 2;
        case '^': return 3;
        default: return 0;
    }
}


int isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/'|| c=='^';
}


void infixToPostfix(const char *infix, char *postfix) {
    int i = 0, j = 0;
    while (infix[i]) {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(infix[i]);
        } else if (infix[i] == ')') {
            while (top != -1 && peek() != '(') {
                postfix[j++] = pop();
            }
            pop();
        } else if (isOperator(infix[i])) {
            while (top != -1 && precedence(peek()) >= precedence(infix[i])) {
                postfix[j++] = pop();
            }
            push(infix[i]);
        }
        i++;
```

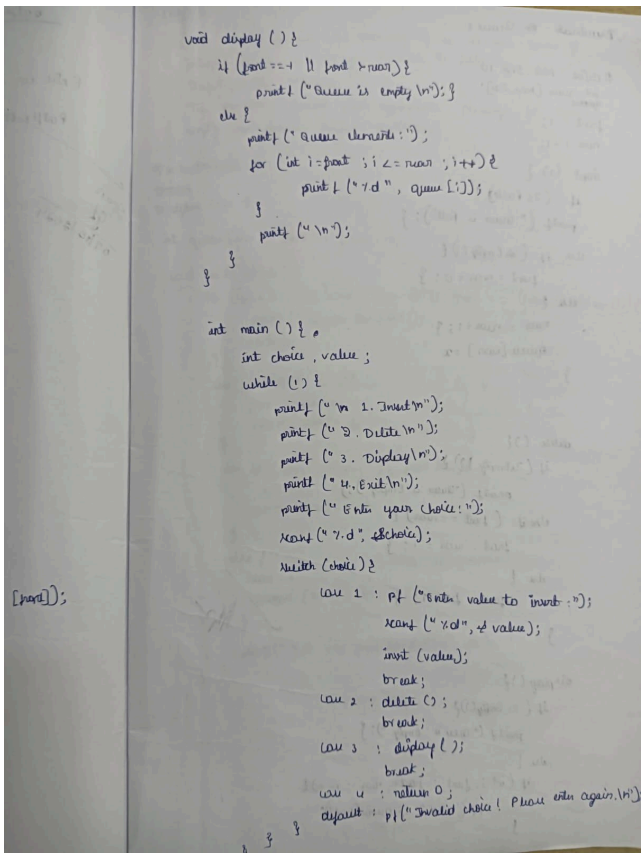
```
    }
    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];

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

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

    return 0;
}
```

```
Enter an infix expression: a+b*(c^d-e)^(f+g*h)-i
Postfix expression: abcd^e-fgh+^*+i-
PS C:\Users\Spandana\OneDrive\Documents\dsa programs\output>
```

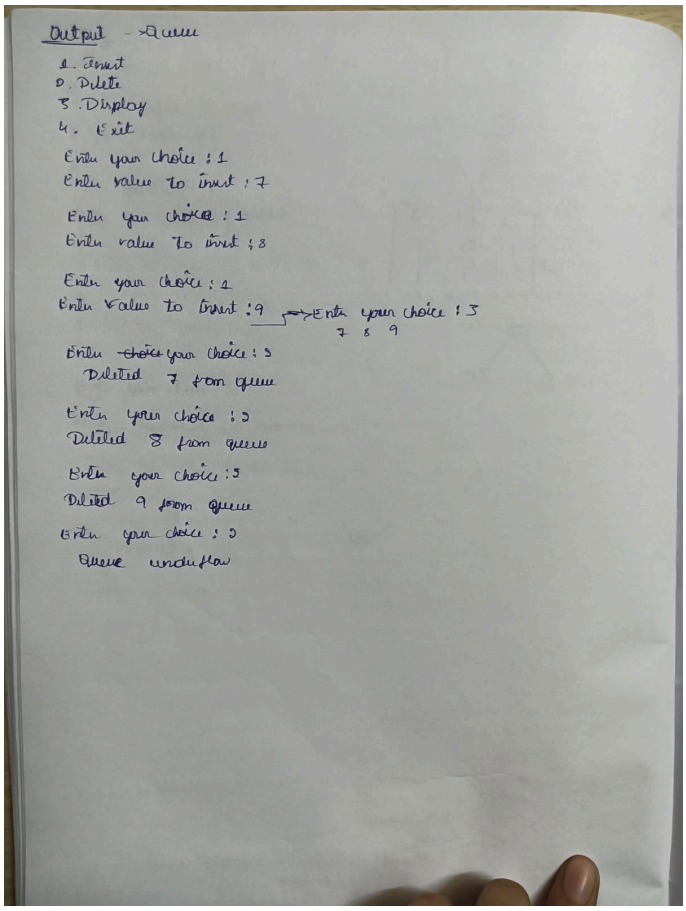
3.3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert,Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions



```
#define MAX_SIZE 10
int queue[MAX_SIZE];
int front = -1;
int rear = -1;
void insert(int value) {
    if(rear == MAX_SIZE-1)
    {
        printf("Queue Overflow ! Cannot insert.\n");
    } else {
        if (front == -1)
        {
            front = 0;
        }
        rear++;
        queue[rear] = value;
        printf("Inserted %d into the queue.\n", value);
        }
}
void delete() {
  if (front == -1 || front > rear) {
        printf("Queue underflow ! Cannot delete \n");
    } else {
        printf("Deleted %d from the queue.\n", queue[front]);
        front++;
        }
    }
```



```
void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty\n"); }
    else {
        printf("Queue elements:");
        for (int i=front; i<=rear; i++) {
            printf("%d", queue[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice, value;
    while(1) {
        printf("\n 1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice) {
[front]);
            case 1: pf("Enter value to insert:");
                    scanf("%d", &value);
                    insert(value);
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: return 0;
            default: pf("Invalid choice! Please enter again.\n");
        }
    }
```




```c
#include<stdio.h>
#include <stdlib.h>
#define size 3

int queue[size];
int front=-1;
int rear=-1;

void insert(int value){
    if(rear==size-1){
        printf("Queue Overflow!\n");
    }
    else{
        front=0;
        rear++;
        queue[rear]=value;
        printf("%d is inserted into queue\n",value);
    }
}
```

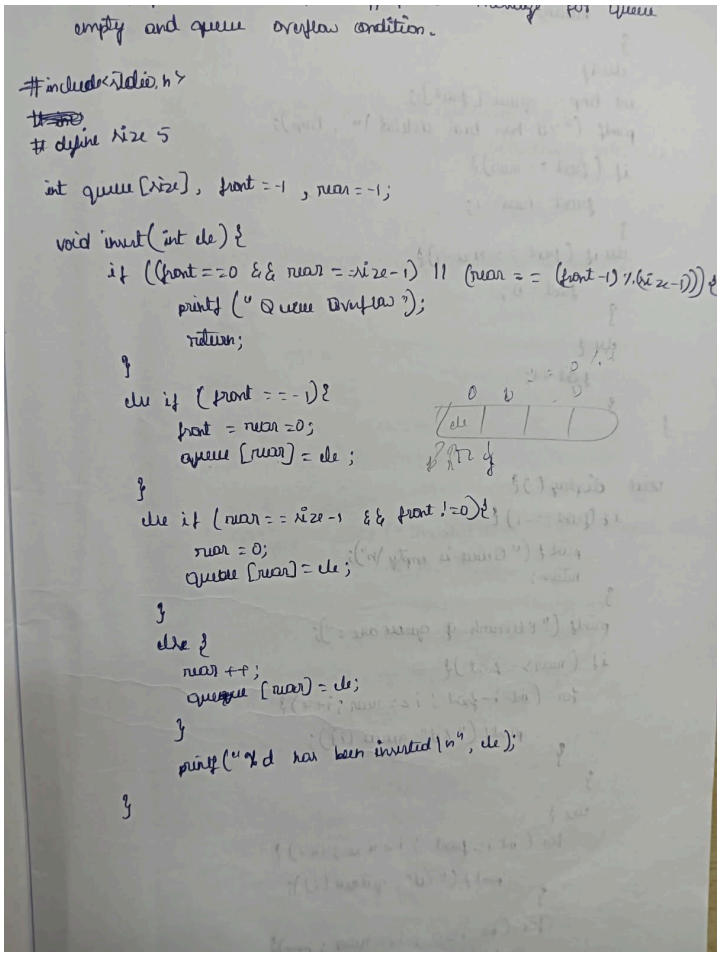
```
void delete(){
    if(front==-1||front>rear){
        printf("Queue Underflow!\n");
    }
    else{
        printf("%d is deleted from the queue\n",queue[front]);
        front++;
    }
}

void display(){
    if(front==-1||front>rear){
        printf("Queue is Empty\n");
    }
    else{
        printf("Queue elements are:");
        for(int i=front;i<=rear;i++){
            printf("%d ",queue[i]);
        }printf("\n");
    }
}

void main(){
    int value;
    int choice;
    while(1){
        printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1:printf("Enter a value to insert:");
            scanf("%d",&value);
            insert(value);
            break;
            case 2:delete();
            break;
            case 3:display();
            break;
            case 4:exit(0);
            default:printf("Invalid choice!Please enter again");
```

```
            }

        }

}
```

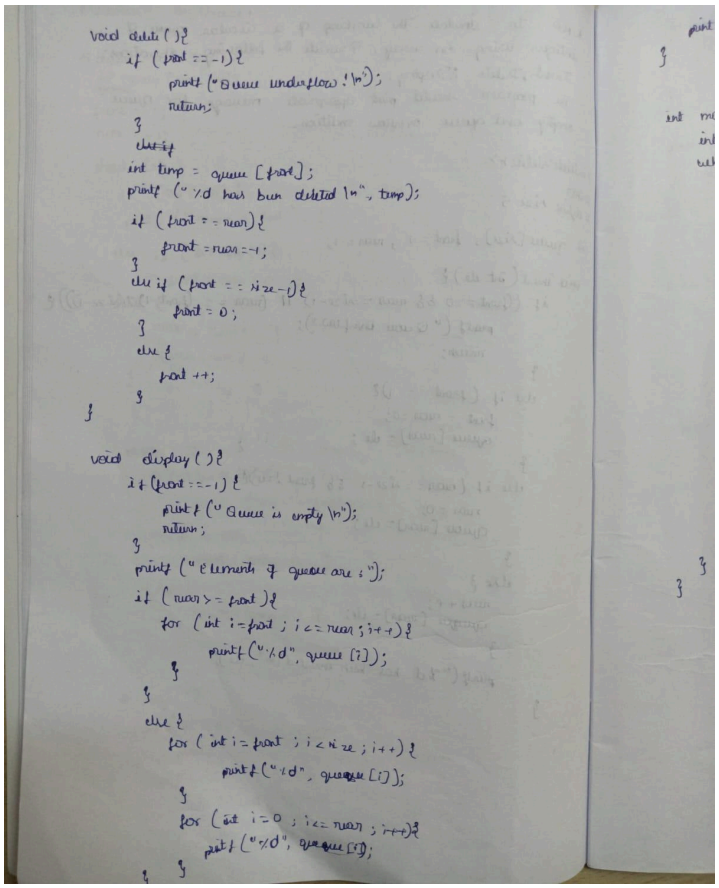
```
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter a value to insert:2
2 is inserted into queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter a value to insert:4
4 is inserted into queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter a value to insert:6
6 is inserted into queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter a value to insert:8
Queue Overflow!
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:3
Queue elements are:2 4 6
1.Insert
2.Delete
3.Display
4.Exit
```

```
Enter your choice:2
2 is deleted from the queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:2
4 is deleted from the queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:2
6 is deleted from the queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:2
Queue Underflow!
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:3
Queue is Empty
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:4
PS C:\Users\Spandana\OneDrive\Documents\dsa programs\output>
```

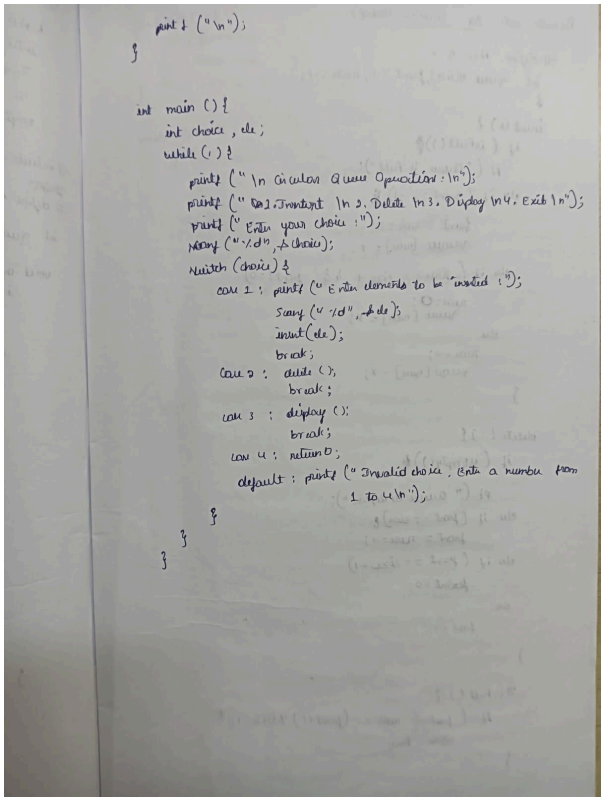
3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions



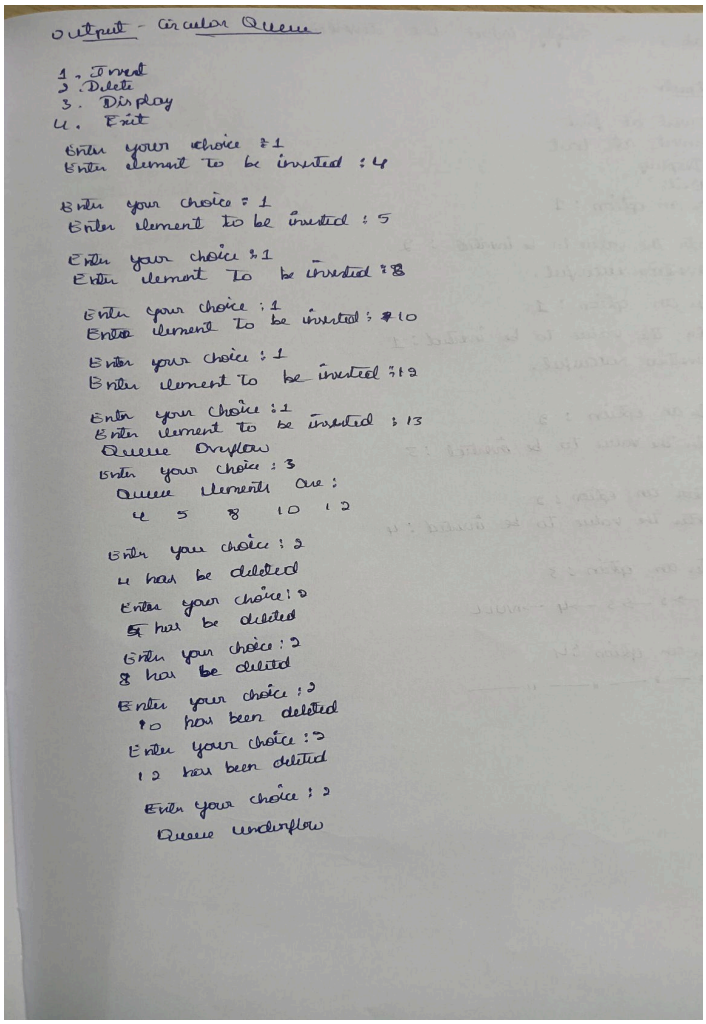
```
empty and queue overflow condition.

#include<stdio.h>
#define size 5
int queue[size], front = -1, rear = -1;
void insert(int de){
    if ((front==0 && rear==size-1) || (rear == (front-1)%(size-1))){
        printf("Queue Overflow");
        return;
    }
    else if (front==-1){
        front = rear=0;
        queue[rear] = de;
    }
    else if (rear==size-1 && front!=0){
        rear = 0;
        queue[rear] = de;
    }
    else{
        rear++;
        queue[rear] = de;
    }
    printf("%d has been inserted\n", de);
}
```



```
void delete ( ){
    if ( front == -1 ){
        printf ("Queue underflow !\n");
        return;
    }
    else
    int temp = queue [front];
    printf ("%d has been deleted \n", temp);
    if ( front == rear ){
        front = rear = -1;
    }
    else if ( front == size-1 ){
        front = 0;
    }
    else {
        front ++;
    }
}

void display ( ){
    if (front == -1 ) {
        printf ("Queue is empty \n");
        return;
    }
    printf ("Elements of queue are : ");
    if ( rear >= front ){
        for ( int i=front ; i<= rear ; i++ ){
            printf ("%d", queue [i]);
        }
    }
    else {
        for ( int i= front ; i< size ; i++ ){
            printf ("%d", queue [i]);
        }
        for ( int i=0 ; i<= rear ; i++){
            printf ("%d", queue [i]);
        }
    }
```



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

int main (){
    int choice , ele;
    while (1) {
        printf ("\n Circular Queue Operations :\n");
        printf ("1. Insert \n 2. Delete \n 3. Display \n 4. Exit \n");
        printf ("Enter your choice :");
        scanf ("%d", &choice);
        switch (choice) {
            case 1 : printf ("Enter elements to be inserted :");
                     scanf ("%d", &ele);
                     insert(ele);
                     break;
            case 2 : delete ();
                     break;
            case 3 : display ();
                     break;
            case 4 : return 0;
            default : printf ("Invalid choice. Enter a number from
                               1 to 4\n");
        }
    }
}
```




```c
#include<stdio.h>
#define size 3
int queue [size],front=-1,rear=-1;
void insert (int ele)
{
    if ((front== 0 && rear== size -1) || (rear == (front-1)%(size-1)))
    {
        printf("queue overflow\n");
        return;
    }
    else if (front ==-1)
    {
        front=rear=0;
        queue[rear]=ele;
    }
    else if ((rear == size-1 )&& (front=!0))
```

```
    {
        rear=0;
        queue [rear] =ele;
    }
    else
    {
        rear++;
        queue[rear]= ele;
    }
    printf(" %d has been inserted\n", ele);
}
void delete()
{
    if (front == -1)
    {
        printf("queue underflow\n");
        return;
    }
    int temp = queue[front];
    printf("%d has been deleted", temp);
    if (front== rear)
    {
        front= rear= -1;

    }
    else if (front== size-1)
    {
        front =0;
    }
    else
    {
        front ++;
    }
}
void display()
{
    if (front==-1)
    {
        printf("queue is empty");
        return;
    }
```

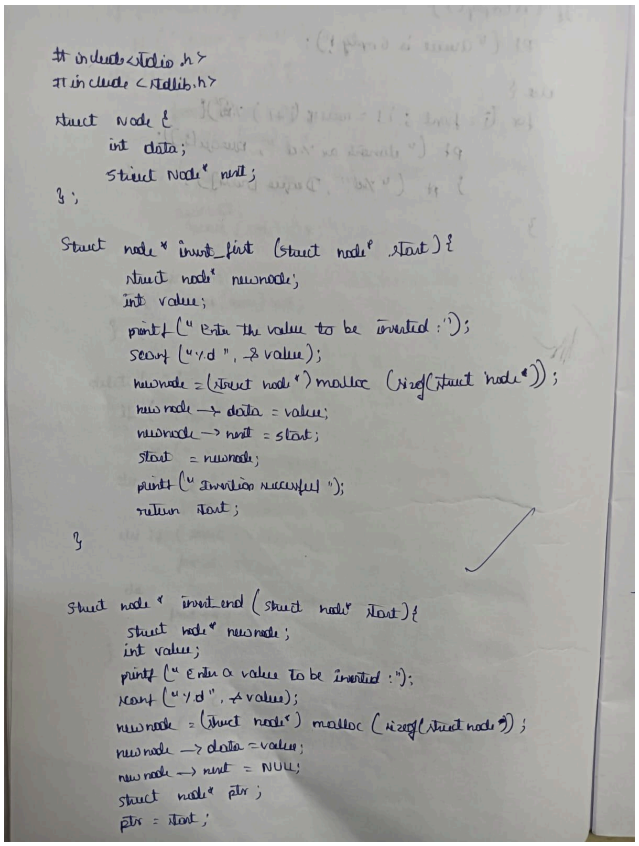
```
    printf("elements of the queue are:");
    if(rear>=front)
    {
        for(int i= front; i<= rear;i++)
        {
            printf("%d ", queue[i]);
        }
    }
    else
    {
        for(int i= front; i< size; i++)
        {
            printf("%d ", queue[i]);
        }
        for(int i=0; i<= rear;i++)
        {
            printf("%d ", queue[i]);
        }
    }
    printf("\n");
}
int main()
{
    int choice,ele;
    while (1)
    {
        printf("\n circular queue operators:\n");
        printf("1.insert\n2.delete\n3.display\n4.exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("enter element to be inserted ");
                    scanf("%d",&ele);
                    insert (ele);
                    break;
            case 2: delete ();
                    break;
            case 3: display();
                    break;
            case 4: return 0;
```

```
            default: printf("invalid choice");
        }
    }
}
```

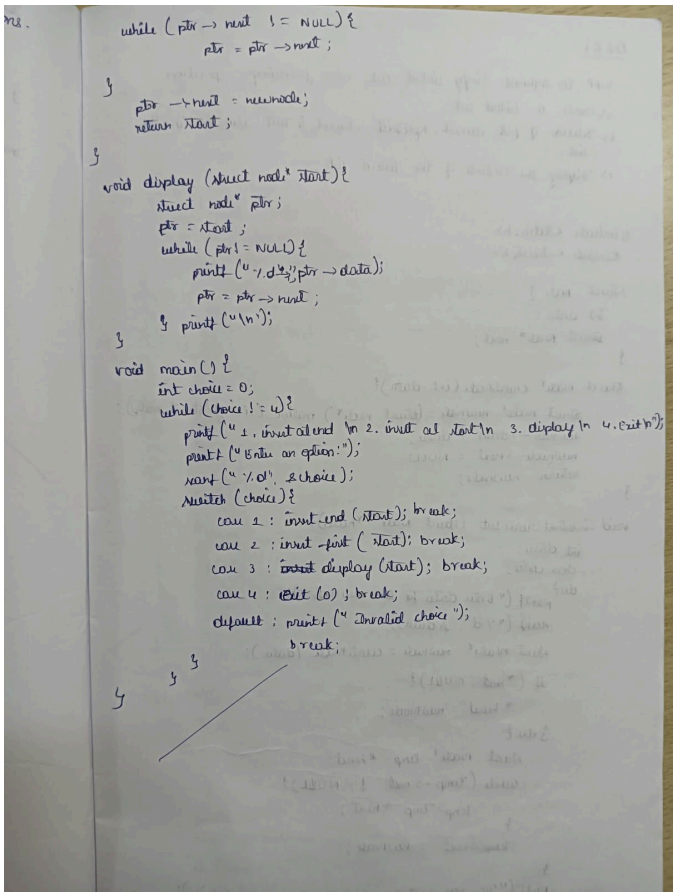
```
 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:1
enter element to be inserted 5
 5 has been inserted

 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:1
enter element to be inserted 6
 6 has been inserted

 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:1
enter element to be inserted 7
 7 has been inserted

 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:1
enter element to be inserted 8
queue overflow
```

```
 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:3
elements of the queue are:5 6 7

 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:2
5 has been deleted
 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:2
6 has been deleted
 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:2
7 has been deleted
 circular queue operators:
1.insert
2.delete
3.display
4.exit
Enter your choice:2
queue underflow
```

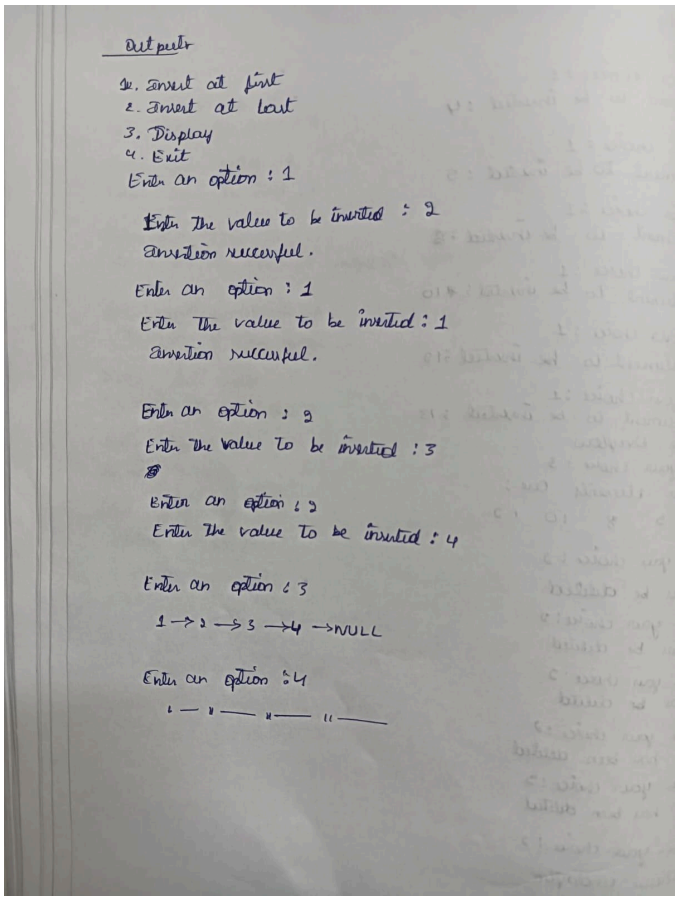
4.WAP to Implement Singly Linked List with following operations
a) Createalinkedlist.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.



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

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

Struct node* insert_first (struct node* start) {
    struct node* newnode;
    int value;
    printf ("Enter the value to be inserted :");
    scanf ("%d", &value);
    newnode = (struct node*) malloc (sizeof(struct node*));
    newnode -> data = value;
    newnode -> next = start;
    start = newnode;
    printf ("Insertion successful");
    return start;
}

Struct node* insert_end (struct node* start) {
    struct node* newnode;
    int value;
    printf ("Enter a value to be inserted :");
    scanf ("%d", &value);
    newnode = (struct node*) malloc (sizeof(struct node*));
    newnode -> data = value;
    newnode -> next = NULL;
    struct node* ptr;
    ptr = start;
```



```
while (ptr -> next != NULL) {
        ptr = ptr -> next;
}
    ptr -> next = newnode;
    return start;
}
void display (struct node* start) {
    struct node* ptr;
    ptr = start;
    while (ptr != NULL) {
        printf ("%d ", ptr -> data);
        ptr = ptr -> next;
    } printf ("\n");
}
void main () {
    int choice = 0;
    while (choice != 4) {
        printf ("1. insert at end \n 2. insert at start \n 3. display \n 4. exit \n");
        printf ("Enter an option:");
        scanf ("%d", &choice);
        switch (choice) {
            case 1 : insert_end (start); break;
            case 2 : insert_first (start); break;
            case 3 : display (start); break;
            case 4 : exit (0); break;
            default : printf ("Invalid choice");
                      break;
        }
    }
```



Output

1. Insert at first
2. Insert at last
3. Display
4. Exit

Enter an option : 1

Enter the value to be inserted : 2

Insertion successful.

Enter an option : 1

Enter the value to be inserted : 1

Insertion successful.

Enter an option : 2

Enter the value to be inserted : 3

Enter an option : 2

Enter the value to be inserted : 4

Enter an option : 3

1 -> 2 -> 3 -> 4 -> NULL

Enter an option : 4

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

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

struct node* insert_begin(struct node *start){
    struct node *newnode;
    int value;
    printf("enter a value to be inserted: ");
    scanf("%d",&value);
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->next = start;
    start = newnode;
    return start;
}
```

```c
struct node* insert_end(struct node *start){
    struct node *newnode, *ptr;
    int value;
    printf("enter a value to be inserted: ");
    scanf("%d",&value);
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->next = NULL;
    ptr = start;
    while(ptr->next != NULL){
        ptr = ptr->next;
    }
    ptr->next = newnode;
    return start;
}

void display(struct node *start){
    struct node *ptr = start;
    printf("the contents of the linked list are: ");
    while(ptr->next != NULL){
        printf("%d -> ",ptr->data);
        ptr=ptr->next;
    }
    printf("%d\n",ptr->data);
}

int main(){
    struct node *start;
    start = (struct node*)malloc(sizeof(struct node));
    int value;
    printf("enter a value to insert at the start: ");
    scanf("%d",&value);
    start->data = value;
    start->next = NULL;
    int choice=0;
    while(choice!=4){
        printf("\n1.INSERT BACK\n2.INSERT FRONT\n3.DISPLAY\n4.EXIT\n");
        printf("enter an option: ");
        scanf("%d",&choice);
        switch(choice){
```

```
            case 1: start = insert_end(start);
            break;
            case 2: start = insert_begin(start);
            break;
            case 3: display(start);
            break;
            case 4: break;
            default: printf("enter a valid option!!\n");
        }
    }
    return 0;
}
```

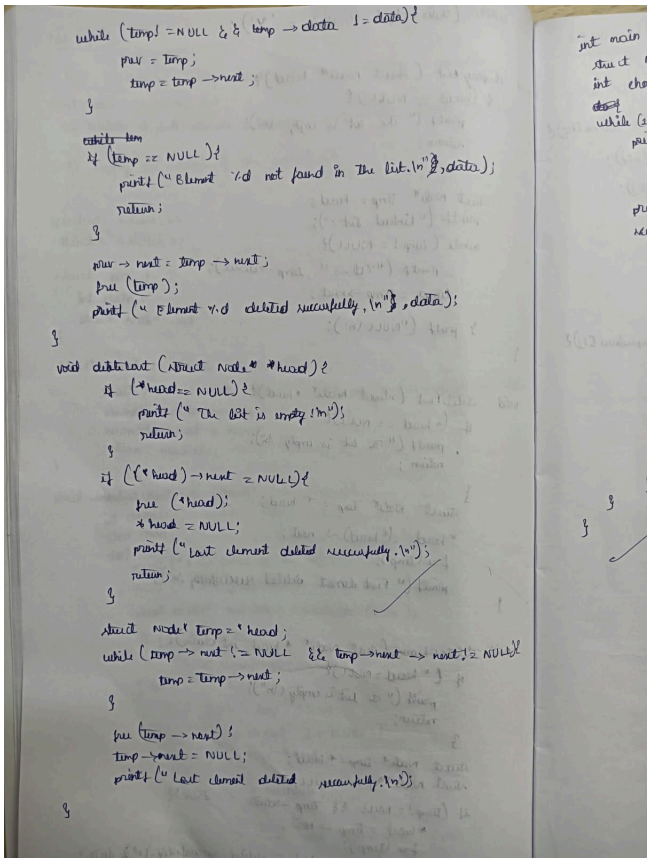
```
enter a value to insert at the start: 6

1.INSERT BACK
2.INSERT FRONT
3.DISPLAY
4.EXIT
enter an option: 1
enter a value to be inserted: 2

1.INSERT BACK
2.INSERT FRONT
3.DISPLAY
4.EXIT
enter an option: 1
enter a value to be inserted: 4

1.INSERT BACK
2.INSERT FRONT
3.DISPLAY
4.EXIT
enter an option: 2
enter a value to be inserted: 5

1.INSERT BACK
2.INSERT FRONT
3.DISPLAY
4.EXIT
enter an option: 2
enter a value to be inserted: 7

1.INSERT BACK
2.INSERT FRONT
3.DISPLAY
4.EXIT
enter an option: 3
the contents of the linked list are: 7 -> 5 -> 6 -> 2 -> 4
```

5.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.

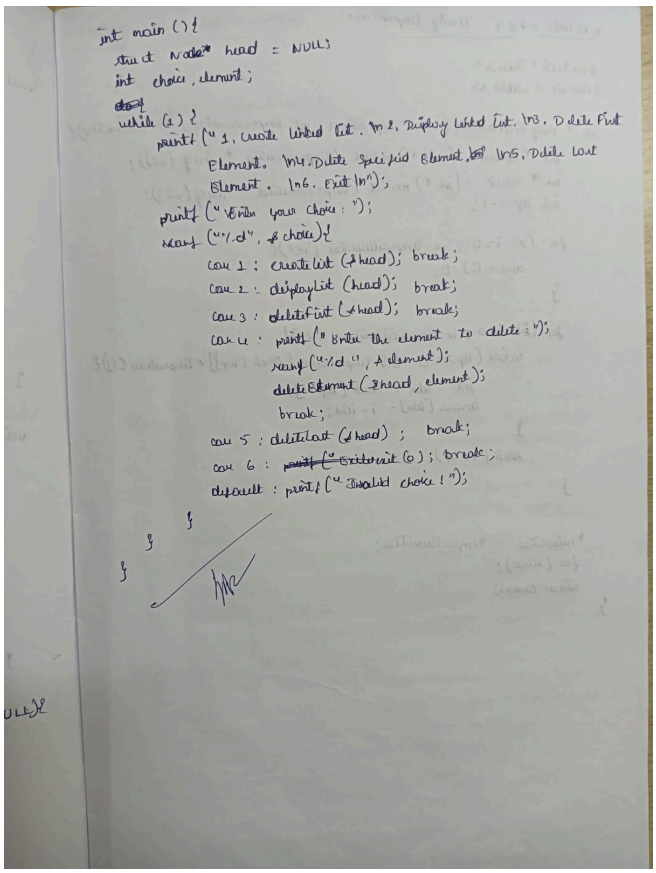
Lab 6 :

Wpp to implement singly linked list with following operation
a> create a linked list.
b> Deletion of first element, specified element & last element in the list.
c> Display the contents of the linked list.

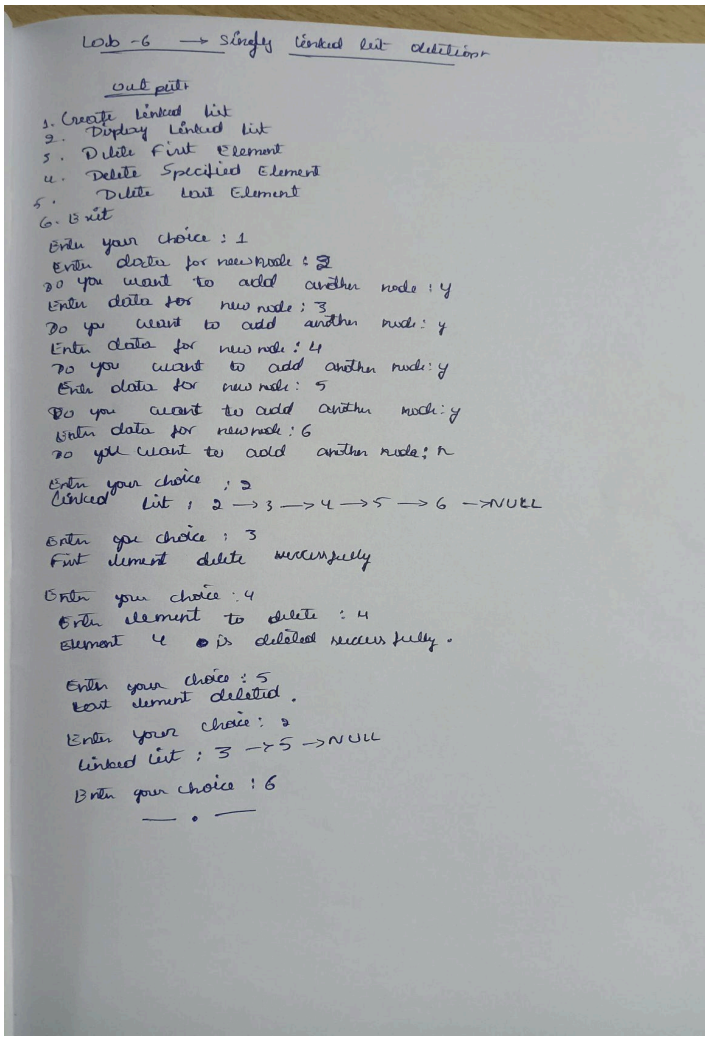
```
#include <stdio.h>
#include <stdlib.h>

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

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

void createList(struct Node** head){
    int data;
    char choice;
    do{
        printf("Enter data for the new node:");
        scanf("%d", &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;
        }
        printf("Do you want to add another node? (y/n):");
        scanf("%c", &choice);
    while (choice == 'y' || choice == 'Y');
}

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

void deleteFirst (struct Node* *head){
    if (*head == NULL){
        printf("The list is empty.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head) -> next;
    free(temp);
    printf("First element deleted successfully.\n");
}

void deleteElement(struct Node* *head, int data){
    if (*head == NULL){
        printf("The list is empty.\n");
        return;
    }
    struct Node* temp = *head;
    struct Node** prev = NULL;
    if (temp != NULL && temp->data == data){
        *head = temp -> next;
        free(temp);
        printf("Element %d is deleted successfully.\n", data);
```



```
while (temp != NULL && temp -> data != data){
    prev = temp;
    temp = temp -> next;
}
if (temp == NULL){
    printf("Element %d not found in the list.\n", data);
    return;
}
prev -> next = temp -> next;
free(temp);
printf("Element %d deleted successfully.\n", data);
}
void deleteLast(struct Node** head){
    if (*head == NULL){
        printf("The list is empty!\n");
        return;
    }
    if ((*head) -> next == NULL){
        free(*head);
        *head = NULL;
        printf("Last element deleted successfully.\n");
        return;
    }
    struct Node* temp = *head;
    while (temp -> next != NULL && temp -> next -> next != NULL){
        temp = temp -> next;
    }
    free(temp -> next);
    temp -> next = NULL;
    printf("Last element deleted successfully.\n");
}
```



```
int main(){
    struct Node* head = NULL;
    int choice, element;
    while (1){
        printf("1. Create Linked List. \n2. Display Linked List. \n3. Delete First
        Element. \n4. Delete Specified Element. \n5. Delete Last
        Element. \n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        case 1: createList(&head); break;
        case 2: displayList(head); break;
        case 3: deleteFirst(&head); break;
        case 4: printf("Enter the element to delete: ");
                scanf("%d", &element);
                deleteElement(&head, element);
                break;
        case 5: deleteLast(&head); break;
        case 6: exit(0); break;
        default: printf("Invalid choice!");
    }
}
```



Lab - 6 → Singly linked list deletion

output

1. Create linked list
2. Display linked list
3. Delete First Element
4. Delete Specified Element
5. Delete last Element
6. Exit

Enter your choice : 1
Enter data for new node : 2
Do you want to add another node : y
Enter data for new node : 3
Do you want to add another node : y
Enter data for new node : 4
Do you want to add another node : y
Enter data for new node : 5
Do you want to add another node : y
Enter data for new node : 6
Do you want to add another node : n

Enter your choice : 2
Linked list : 2 → 3 → 4 → 5 → 6 → NULL

Enter your choice : 3
First element delete successfully

Enter your choice : 4
Enter element to delete : 4
Element 4 is deleted successfully.

Enter your choice : 5
Last element deleted.

Enter your choice : 2
Linked list : 3 → 5 → NULL

Enter your choice : 6

```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));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return NULL;
    }
```

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


void createList(struct Node** head) {
    int data;
    char choice;
    do {
        printf("Enter data for the new node: ");
        scanf("%d", &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;
        }

        printf("Do you want to add another node? (y/n): ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');
}


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

    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
```

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


void deleteFirst(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty, cannot delete the first element.\n");
        return;
    }

    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("First element deleted successfully.\n");
}


void deleteElement(struct Node** head, int data) {
    if (*head == NULL) {
        printf("The list is empty, cannot delete the element.\n");
        return;
    }

    struct Node* temp = *head;
    struct Node* prev = NULL;


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


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

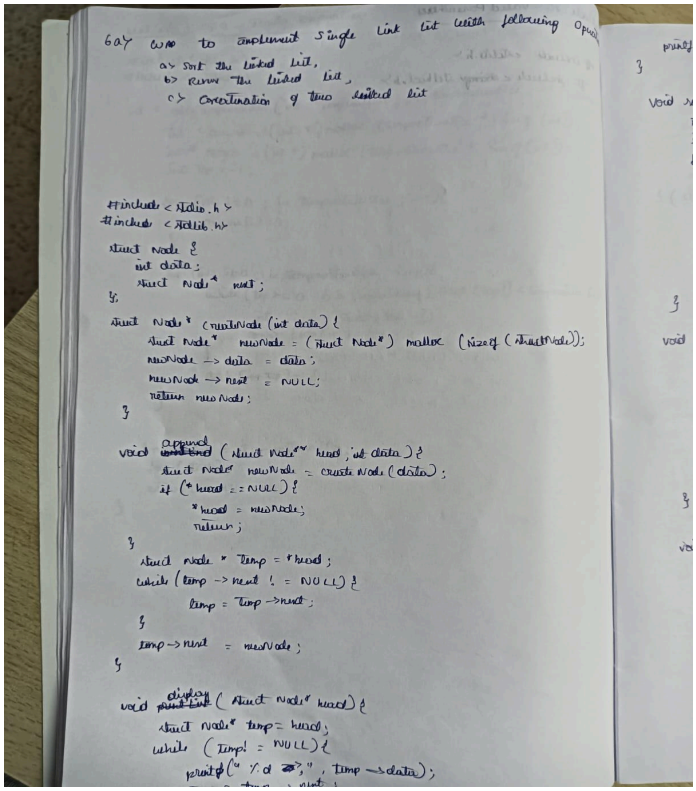
```
    }


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

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


void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty, cannot delete the last element.\n");
        return;
    }


    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        printf("Last element deleted successfully.\n");
        return;
    }

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


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

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

```
    int choice, element;

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

        switch (choice) {
            case 1:
                createList(&head);
                break;
            case 2:
                displayList(head);
                break;
            case 3:
                deleteFirst(&head);
                break;
            case 4:
                printf("Enter the element to delete: ");
                scanf("%d", &element);
                deleteElement(&head, element);
                break;
            case 5:
                deleteLast(&head);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

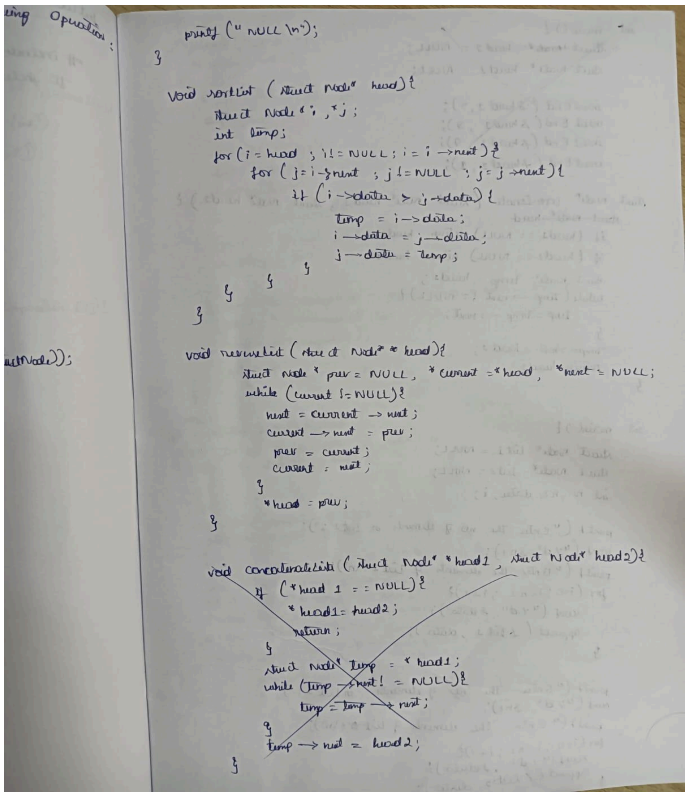
```
Menu:
1. Create Linked List
2. Display Linked List
3. Delete First Element
4. Delete Specified Element
5. Delete Last Element
6. Exit
Enter your choice: 1
Enter data for the new node: 1
Do you want to add another node? (y/n): y
Enter data for the new node: 2
Do you want to add another node? (y/n): y
Enter data for the new node: 3
Do you want to add another node? (y/n): y
Enter data for the new node: 4
Do you want to add another node? (y/n): y
Enter data for the new node: 5
Do you want to add another node? (y/n): y
Enter data for the new node: 6
Do you want to add another node? (y/n): n

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

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

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

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

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

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

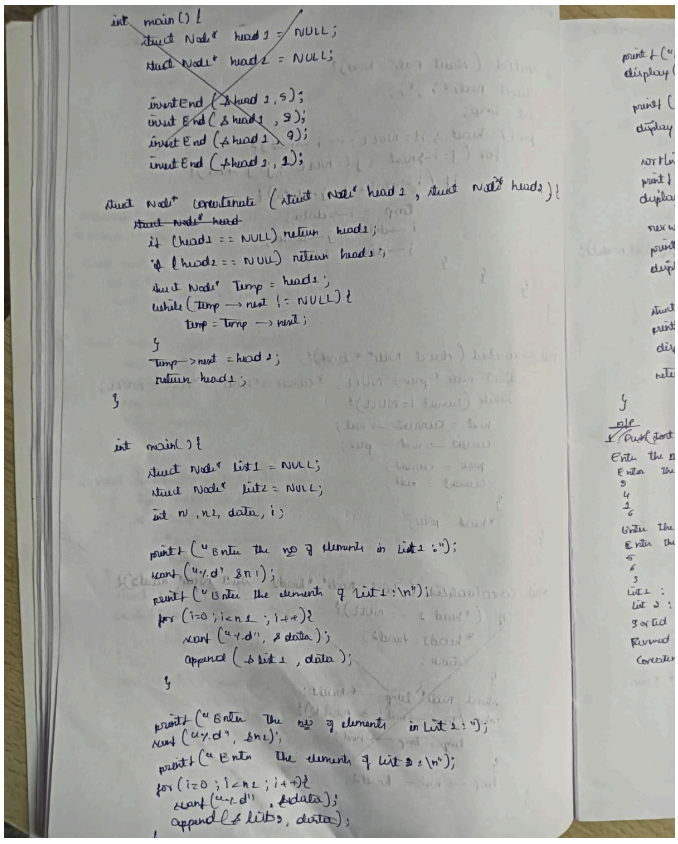
6.6a) WAP to Implement Single Link List with following operations: Sortthelinkedlist, Reversethelinkedlist,Concatenation of two linked lists.



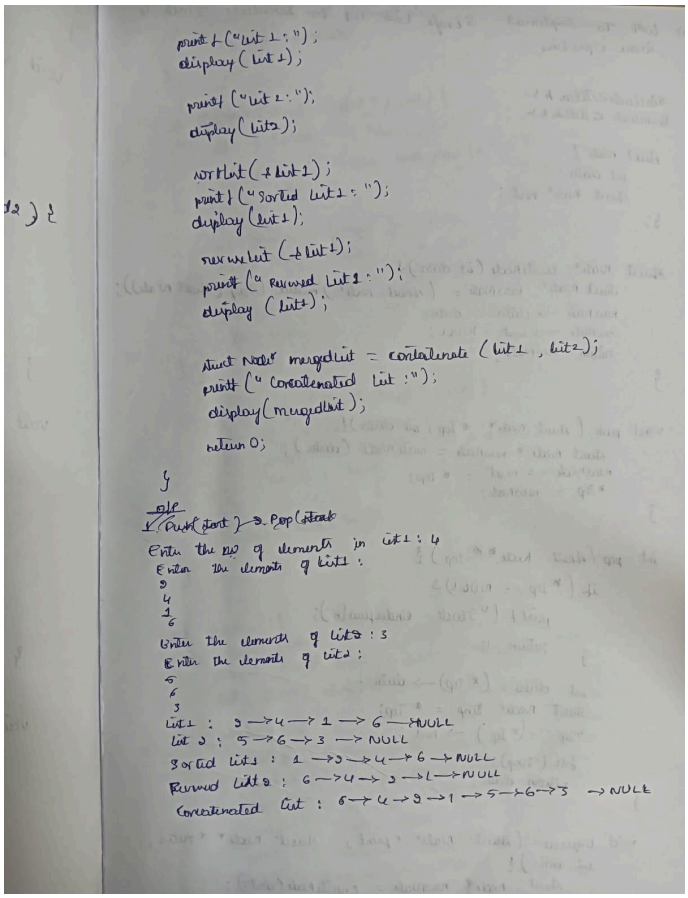
```
6a) WAP to implement single link list with following operations
    a) sort the linked list,
    b) reverse the linked list,
    c) Concatenation of two linked list

#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc (sizeof(struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = newNode;
}
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp -> data);
```



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



```
int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;

    insertEnd(&head1, 9);
    insertEnd(&head1, 9);
    insertEnd(&head1, 9);
    insertEnd(&head1, 1);

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

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int n1, n2, data, i;

    printf("\n Enter the no of elements in list1 :");
    scanf("%d", &n1);
    printf("\n Enter the elements of list1 :\n");
    for (i=0; i<n1; i++) {
        scanf("%d", &data);
        append(&list1, data);
    }

    printf("\n Enter the no of elements in list2 :");
    scanf("%d", &n2);
    printf("\n Enter the elements of list2 :\n");
    for (i=0; i<n2; i++) {
        scanf("%d", &data);
        append(&list2, data);
```



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

    printf("List 2 :");
    display(list2);

    sortList(&list1);
    printf("\n Sorted List1 :");
    display(list1);

    reverseList(&list1);
    printf("\n Reversed List1 :");
    display(list1);

    struct Node* mergedList = concatenate(list1, list2);
    printf("\n Concatenated List :");
    display(mergedList);

    return 0;
}
```

```
Enter the no of elements in list1 : 4
Enter the elements of list1 :
9
4
1
6
Enter the elements of list2 : 3
Enter the elements of list2 :
5
6
3
List 1 : 9 -> 4 -> 1 -> 6 -> NULL
List 2 : 5 -> 6 -> 3 -> NULL
Sorted Lists : 1 -> 4 -> 6 -> 9 -> NULL
Reversed List 1 : 6 -> 4 -> 9 -> 1 -> NULL
Concatenated List : 6 -> 4 -> 9 -> 1 -> 5 -> 6 -> 3 -> NULL
```

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


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

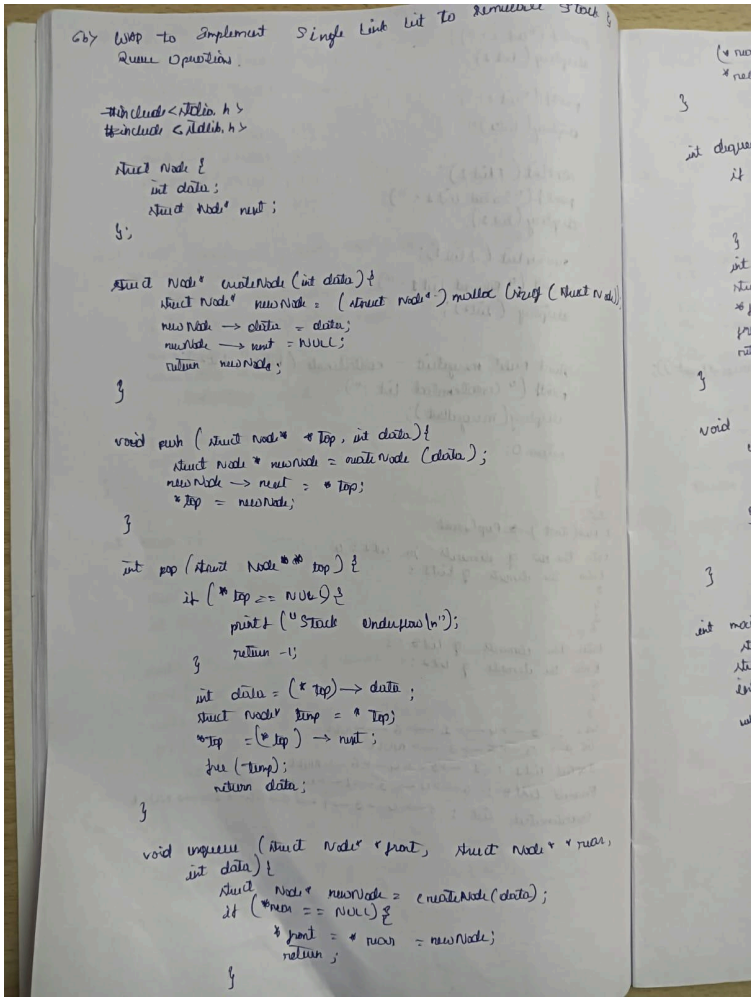
```
void sortList(struct Node** head) {
    struct Node* i, *j;
    int temp;
    for (i = *head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data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


struct Node* concatenate(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) return head2;
    if (head2 == NULL) return head1;

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

int main() {
```

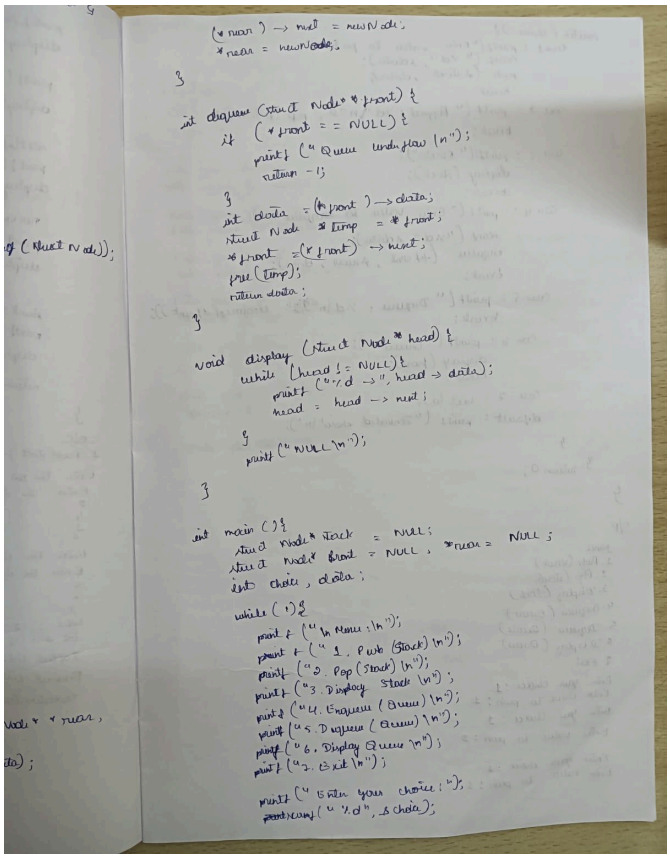
```
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;
    int n1, n2, data, i;


    printf("Enter the number of elements in List 1: ");
    scanf("%d", &n1);
    printf("Enter the elements of List 1:\n");
    for (i = 0; i < n1; i++) {
        scanf("%d", &data);
        append(&list1, data);
    }


    printf("Enter the number of elements in List 2: ");
    scanf("%d", &n2);
    printf("Enter the elements of List 2:\n");
    for (i = 0; i < n2; i++) {
        scanf("%d", &data);
        append(&list2, data);
    }

    printf("List 1: ");
    display(list1);

    printf("List 2: ");
    display(list2);


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


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


    struct Node* mergedList = concatenate(list1, list2);
    printf("Concatenated List: ");
```

```
    display(mergedList);

    return 0;
}
```

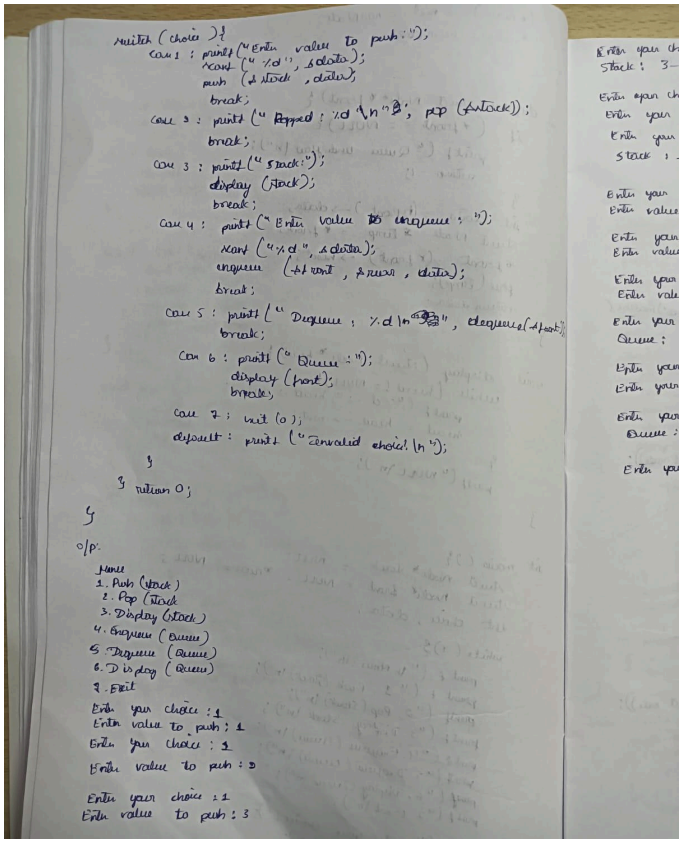
```
Enter the number of elements in List 1: 3
Enter the elements of List 1:
5
4
7
Enter the number of elements in List 2: 3
Enter the elements of List 2:
8
9
10
List 1: 5 -> 4 -> 7 -> NULL
List 2: 8 -> 9 -> 10 -> NULL
Sorted List 1: 4 -> 5 -> 7 -> NULL
Reversed List 1: 7 -> 5 -> 4 -> NULL
Concatenated List: 7 -> 5 -> 4 -> 8 -> 9 -> 10 -> NULL
```

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





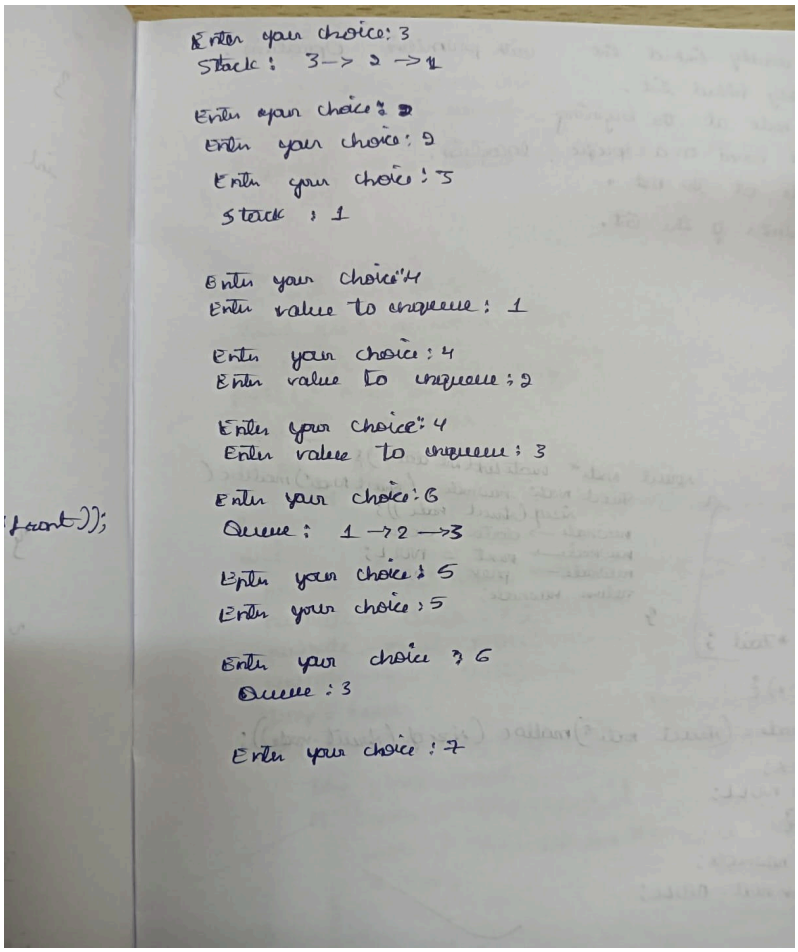
```
    (*rear) -> next = newNode;
    *rear = newNode;
}

int dequeue (struct Node **front) {
    if (*front == NULL) {
        printf ("Queue underflow \n");
        return -1;
    }
    int data = (*front) -> data;
    struct Node *temp = *front;
    *front = (*front) -> next;
    free (temp);
    return data;
}

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

int main () {
    struct Node* stack = NULL;
    struct Node* front = NULL, *rear = NULL;
    int choice, data;
    while (1) {
        printf ("\n Menu :\n");
        printf ("1. Push (Stack) \n");
        printf ("2. Pop (Stack) \n");
        printf ("3. Display Stack \n");
        printf ("4. Enqueue (Queue) \n");
        printf ("5. Dequeue (Queue) \n");
        printf ("6. Display Queue \n");
        printf ("7. Exit \n");
        printf ("Enter your choice : ");
        scanf ("%d", &choice);
```



```
switch (choice) {
    case 1 : printf ("Enter value to push : ");
             scanf ("%d", &data);
             push (&stack, data);
             break;
    case 2 : printf ("Popped : %d \n", pop (&stack));
             break;
    case 3 : printf ("Stack :");
             display (stack);
             break;
    case 4 : printf ("Enter value to enqueue : ");
             scanf ("%d", &data);
             enqueue (&front, &rear, data);
             break;
    case 5 : printf ("Dequeue : %d \n", dequeue(&front));
             break;
    case 6 : printf ("Queue :");
             display (front);
             break;
    case 7 : exit (0);
    default : printf ("Invalid choice! \n");
  }
 return 0;
}
```

o/p:

```
Menu
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display (Queue)
7. Exit
Enter your choice : 1
Enter value to push : 1
Enter your choice : 1
Enter value to push : 2

Enter your choice : 1
Enter value to push : 3
```




```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 push(struct Node** top, int data) {
```

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

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    }
    int data = (*top)->data;
    struct Node* temp = *top;
    *top = (*top)->next;
    free(temp);
    return data;
}


void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(struct Node** front) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }
    int data = (*front)->data;
    struct Node* temp = *front;
    *front = (*front)->next;
    free(temp);
    return data;
}
```

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


int main() {
    struct Node* stack = NULL;
    struct Node *front = NULL, *rear = NULL;

    int choice, data;

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

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

            case 2:
                printf("Popped: %d\n", pop(&stack));
                break;

            case 3:
                printf("Stack: ");
                display(stack);
```

```
                break;

            case 4:
                printf("Enter value to enqueue: ");
                scanf("%d", &data);
                enqueue(&front, &rear, data);
                break;

            case 5:
                printf("Dequeued: %d\n", dequeue(&front));
                break;

            case 6:
                printf("Queue: ");
                display(front);
                break;

            case 7:
                exit(0);

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

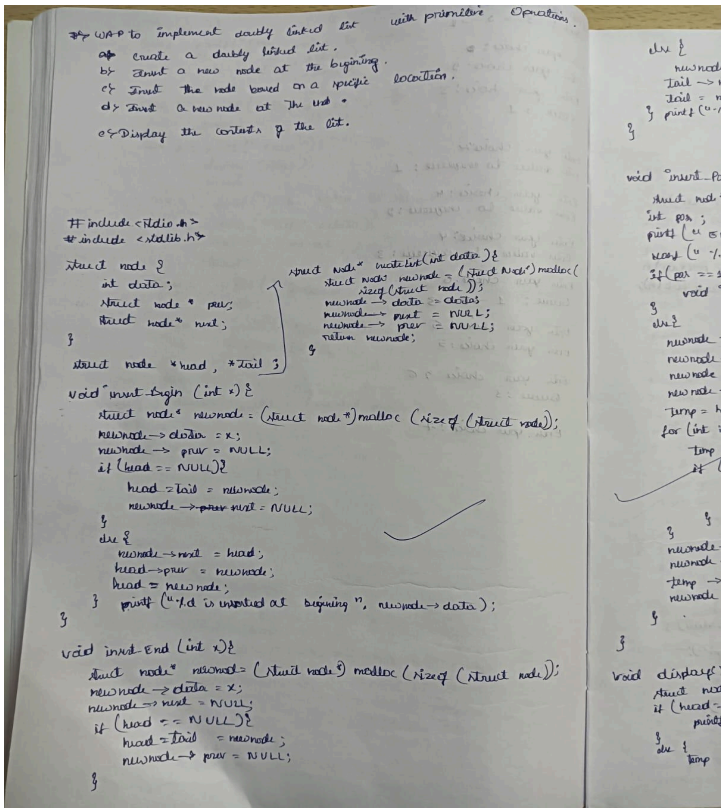
```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push: 1

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push: 2

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push: 3

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

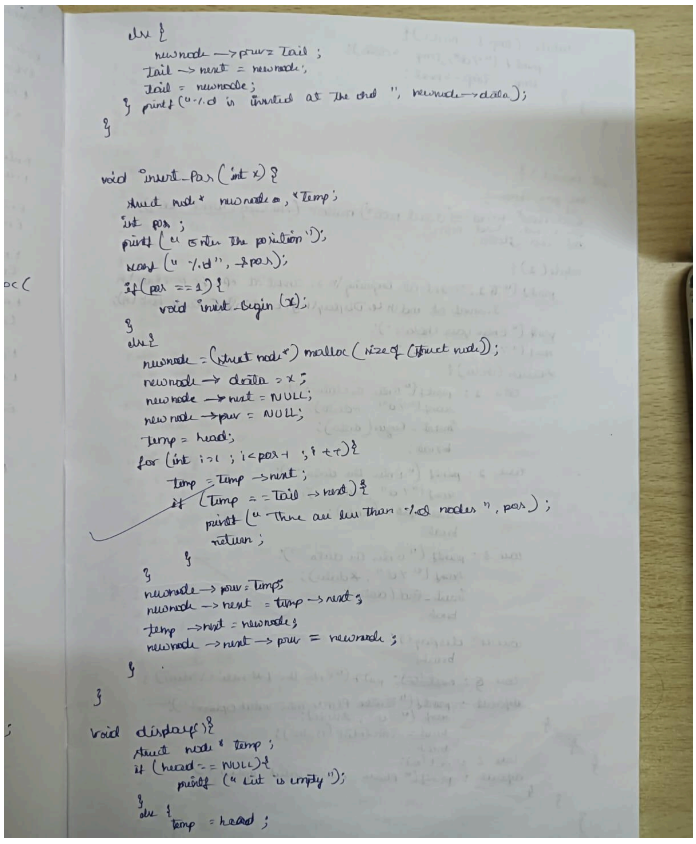
```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 2
Popped: 3

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter value to enqueue: 1

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter value to enqueue: 2

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter value to enqueue: 3
```

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

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 5
Dequeued: 1

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 7
PS C:\Users\Spandana\OneDrive\Documents\dsa programs\output>
```

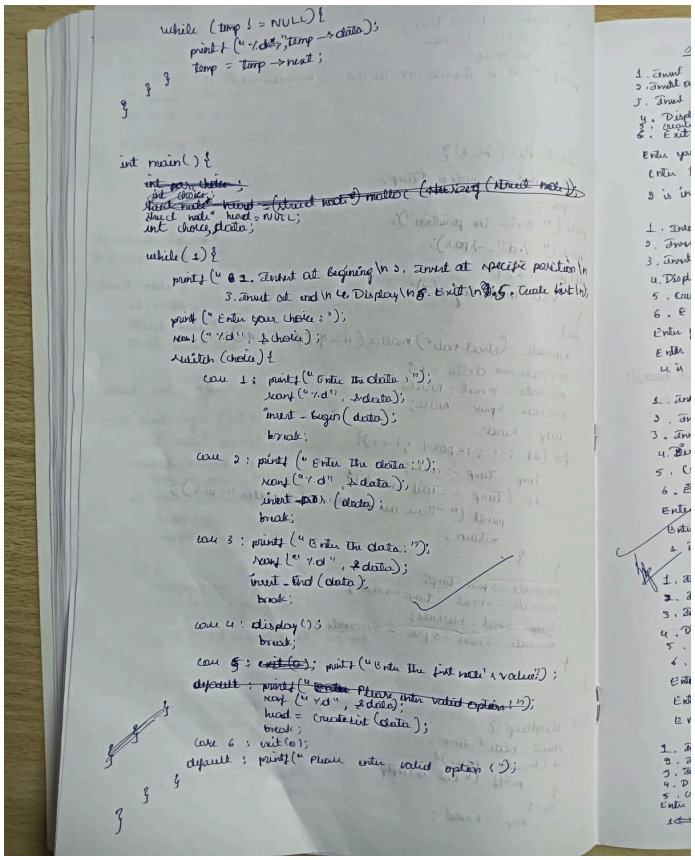
7.WAP to Implement doubly link list with primitive operations

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

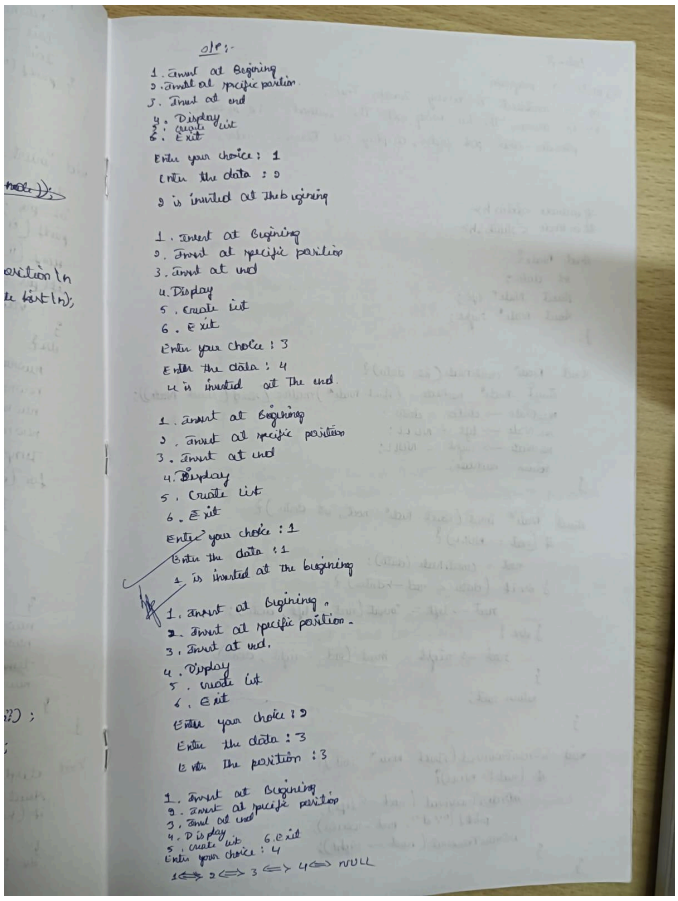




```
    else {
        newnode -> prev = Tail ;
        Tail -> next = newnode;
        Tail = newnode;
    } printf ("%d is inserted at the end ", newnode -> data);
}

void insert_Pos (int x) {
    struct node * newnode, *Temp;
    int pos ;
    printf ("Enter the position");
    scanf ("%d", &pos);
    if(pos == 1) {
        void insert_begin (x);
    }
    else {
    newnode = (struct node*) malloc ( sizeof (struct node));
    newnode -> data = x;
    newnode -> next = NULL;
    newnode -> prev = NULL;
    Temp = head;
    for (int i=1 ; i<pos-1 ; i++) {
        Temp = Temp -> next ;
        if (Temp == 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 ;
    }
}

void display() {
    struct node * temp ;
    if (head == NULL) {
        printf ("List is empty");
    }
    else {
        temp = head ;
```



```
        while (temp != NULL) {
            printf ("%d", temp -> data);
            temp = temp -> next ;
        }
    }
}

int main() {
    int pos, choice;
    struct node* head = (struct node*) malloc (sizeof (struct node));
    struct node* head = NULL;
    int choice, data;

    while (1) {
        printf ("0 1. Insert at beginning\n 2. Insert at specific position\n
                 3. Insert at end\n 4. Display\n 5. Exit\n 6. Create list\n");
        printf ("Enter your choice : ");
        scanf ("%d", &choice);
        switch (choice) {
            case 1 : printf ("Enter the data :");
                     scanf ("%d", &data);
                     insert_begin (data);
                     break;
            case 2 : printf ("Enter the data :");
                     scanf ("%d", &data);
                     insert_pos (data);
                     break;
            case 3 : printf ("Enter the data :");
                     scanf ("%d", &data);
                     insert_end (data);
                     break;
            case 4 : display();
                     break;
            case 5 : exit(0); printf ("Enter the first node's value");
            default : printf ("Please enter valid option !");
                     scanf ("%d", &data);
                     head = CreateList (data);
                     break;
            case 6 : exit(0);
            default : printf ("Please enter valid option ");
        }
    }
}
```




```c
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node* next;
    struct Node* prev;
};struct node *head,*tail;

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

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
```

```
    newNode->prev = NULL;

    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
    printf("Inserted %d at the beginning.\n", data);
}

void insertAtPosition(struct Node** head, int data, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    int count = 1;

    newNode->data = data;

    if (position == 1) {
        newNode->next = *head;
        newNode->prev = NULL;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
        printf("Inserted %d at position %d.\n", data, position);
        return;
    }

    while (temp != NULL && count < position - 1) {
        temp = temp->next;
        count++;
    }

    if (temp == NULL) {
        printf("Position out of bounds.\n");
        return;
    }

    newNode->next = temp->next;
    newNode->prev = temp;

    if (temp->next != NULL) {
```

```
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    printf("Inserted %d at position %d.\n", data, position);
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        printf("List was empty. Inserted %d as the first node.\n", data);
        return;
    }

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

    temp->next = newNode;
    newNode->prev = temp;
    printf("Inserted %d at the end.\n", data);
}

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

    printf("Doubly Linked List contents:\n");
    while (temp != NULL) {
        printf("%d <=> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
```

```
}

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

    while(1){
        printf("\nMenu:\n");
        printf("1. Create the doubly linked list\n");
        printf("2. Insert a new node at the beginning\n");
        printf("3. Insert a new node at a specific position\n");
        printf("4. Insert a new node at the end\n");
        printf("5. Display the contents of the list\n");
        printf("6. Exit\n");

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

        switch (choice) {
            case 1:
                printf("Enter the first node's value: ");
                scanf("%d", &data);
                head = createList(data);
                printf("List created with first node having value: %d\n",
data);
                break;

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

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

```
            case 4:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 5:
                displayList(head);
                break;

            case 6:
                exit(0);


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

    }

    return 0;
}
```

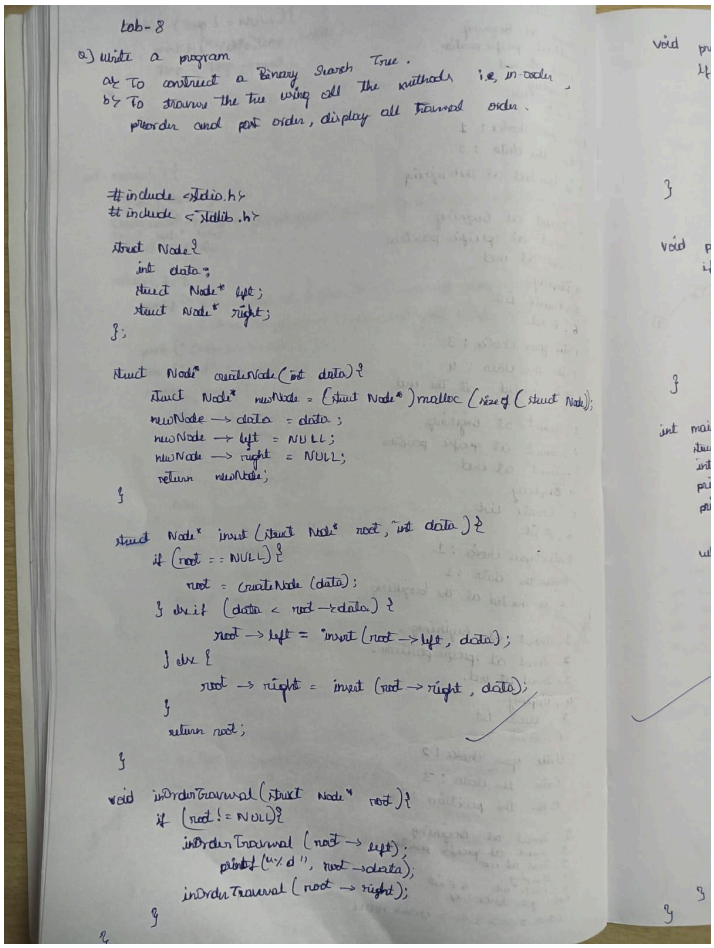
```
Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 1
Enter the first node's value: 2
List created with first node having value: 2

Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 1
Enter the first node's value: 5
List created with first node having value: 5

Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 1
Enter the first node's value: 9
List created with first node having value: 9

Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 1
Enter the first node's value: 14
List created with first node having value: 14
```

```
Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 2
Enter the value to insert at the beginning: 3
Inserted 3 at the beginning.

Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 3
Enter the value to insert: 7
Enter the position: 4
Position out of bounds.

Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 4
Enter the value to insert at the end: 18
Inserted 18 at the end.
```

```
Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 5
Doubly Linked List contents:
3 <=> 14 <=> 18 <=> NULL

Menu:
1. Create the doubly linked list
2. Insert a new node at the beginning
3. Insert a new node at a specific position
4. Insert a new node at the end
5. Display the contents of the list
6. Exit
Enter your choice: 6
```

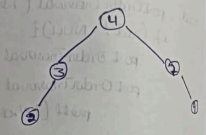
8.Write a program

a) ToconstructabinarySearchtree.
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.



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

void postOrderTraversal (struct Node* root){
    if (root != NULL){
        postOrderTraversal (root -> left);
        postOrderTraversal (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\n2. In-Order Traversal\n3. Pre-order
        Traversal\n4. Post-Order Traversal\n5. 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 :");
                     inOrderTraversal (root);
                     printf ("\n");
                     break;
            case 3 : printf ("Pre-Order Traversal :");
                     preOrderTraversal (root);
                     printf ("\n");
                     break;
            case 4 : printf ("Post-order Traversal:");
                     postOrderTraversal (root); printf ("\n"); break;
            case 5 : exit (0);
            default : printf ("Invalid choice !");
        }
    }
    return 0;
}
```

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

Enter your choice : 1
Enter the value to insert : 4

Enter your choice : 1
Enter the value to insert : 7

Enter your choice : 1
Enter the value to insert : 3

Enter your choice : 1
Enter the value to insert : 9

Enter your choice : 1
Enter the value to insert : 2

Enter your choice : 2
In-order Traversal : 2 3 4 7 9

Enter your choice : 3
Pre-Order Traversal : 4 3 2 7 9

Enter your choice : 4
Post-order Traversal : 2 3 9 7 4

Enter your choice : 5
```



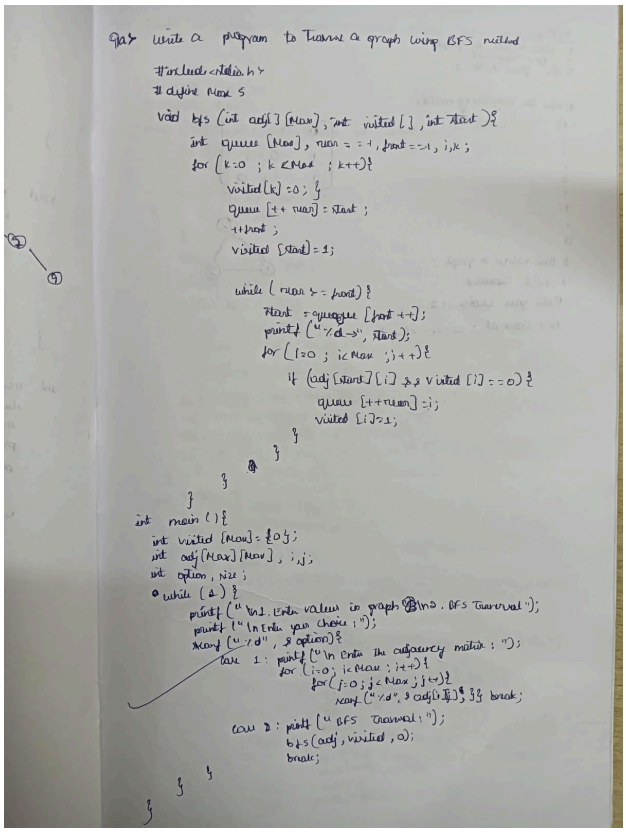
```c
#include <stdio.h>
#include <stdlib.h>


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


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

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


void postOrderTraversal(struct Node* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(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. In-order Traversal\n");
    printf("3. Pre-order Traversal\n");
    printf("4. 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: ");
```

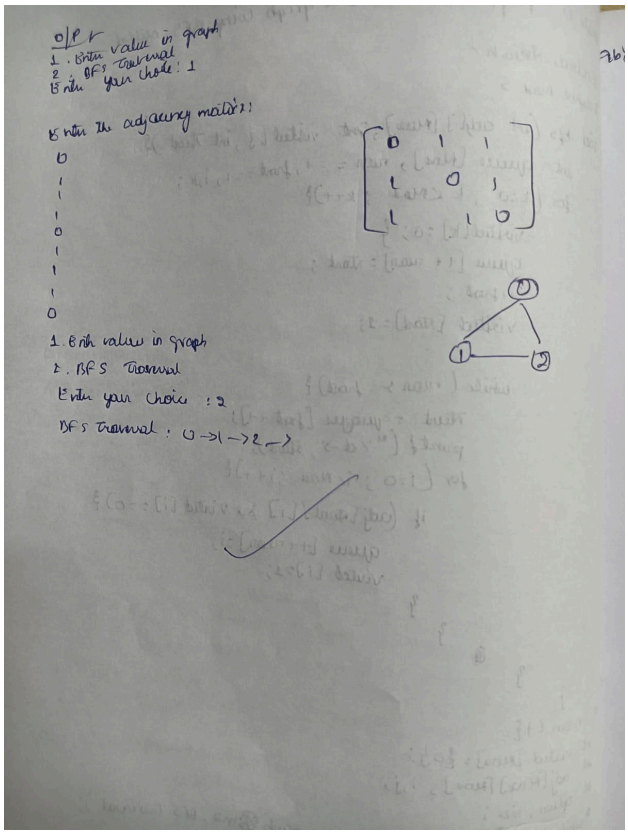
```c
                inOrderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Pre-order Traversal: ");
                preOrderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Post-order Traversal: ");
                postOrderTraversal(root);
                printf("\n");
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

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

Enter your choice: 1
Enter the value to insert: 67

Enter your choice: 1
Enter the value to insert: 45

Enter your choice: 1
Enter the value to insert: 90

Enter your choice: 1
Enter the value to insert: 50

Enter your choice: 1
Enter the value to insert: 72

Enter your choice: 2
In-order Traversal: 45 50 67 72 90

Enter your choice: 3
Pre-order Traversal: 67 45 50 90 72

Enter your choice: 4
Post-order Traversal: 50 45 72 90 67

Enter your choice: 5
```

9.9a) Write a program to traverse a graph using BFS method.



```
Q9a) Write a program to Traverse a graph using BFS method

#include<stdio.h>
#define Max 5
void bfs(int adj[][Max], int visited[], int start){
    int queue[Max], rear=-1, front=-1, i,k;
    for(k=0; k<Max; k++){
        visited[k]=0;}
    queue[++rear]=start;
    ++front;
    visited[start]=1;

    while(rear>=front){
        start=queue[front++];
        printf("%d->", start);
        for(i=0; i<Max;i++){
            if(adj[start][i]&&visited[i]==0){
                queue[++rear]=i;
                visited[i]=1;
            }
        }
    }
}
int main(){
    int visited[Max]={0};
    int adj[Max][Max], i,j;
    int option, size;
    while(1){
        printf("\n1. Enter values in graph\n2. BFS Traversal");
        printf("\nEnter your choice:");
        scanf("%d", &option);
        switch(option){
            case 1: printf("\n Enter the adjacency matrix:");
                    for(i=0; i<Max; i++){
                        for(j=0;j<Max;j++){
                            scanf("%d", &adj[i][j]);}}break;
            case 2: printf("BFS Traversal:");
                    bfs(adj, visited, 0);
                    break;
        }
    }
}
```



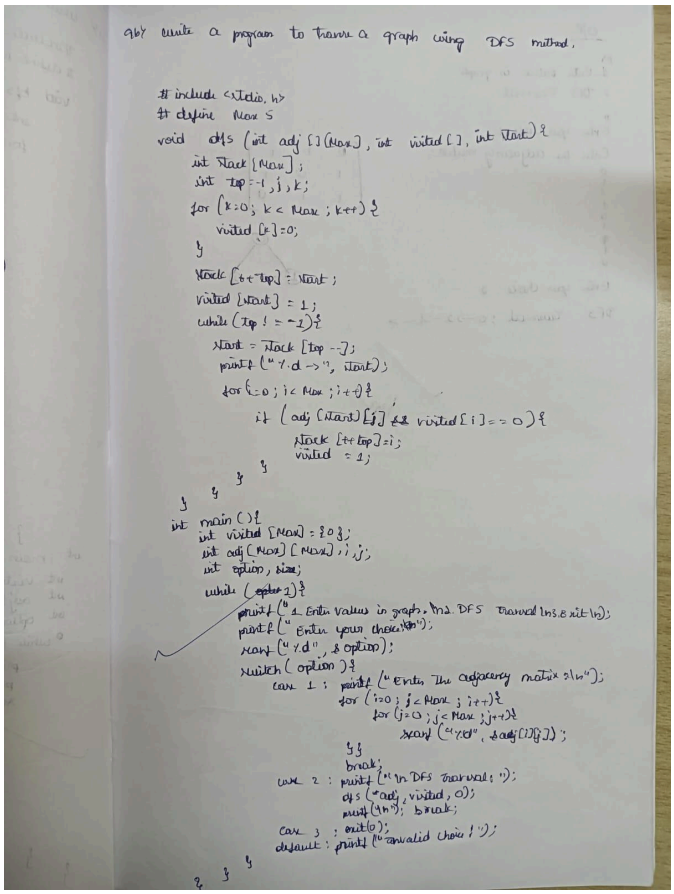
```
O/P:
1. Enter values in graph
2. BFS Traversal
Enter your choice: 1
Enter the adjacency matrix:
0
1
1
1
0
1
1
1
0
1. Enter values in graph
2. BFS Traversal
Enter your choice: 2
BFS Traversal: 0->1->2->
```

```c
#include<stdio.h>
#define MAX 4
void bfs(int adj[][MAX],int visited[],int start)
{
    int queue[MAX],rear=-1,front=-1,i,k;
    for(k=0;k<MAX;k++)
    visited[k]=0;
    queue[++rear]=start;
    ++front;
    visited[start]=1;

    while(rear>=front)
    {
        start=queue[front++];
        printf("%d->",start);
        for(i=0;i<MAX;i++)
        {
            if(adj[start][i]&&visited[i]==0)
            {
                queue[++rear]=i;
                visited[i]=1;
            }
        }
    }
}


int main()
{
    int visited[MAX]={0};
    int adj[MAX][MAX],i,j;
    int option,size;
    do{
        printf("\n *********Main Menu************\n");
        printf("\n 1. Enter values in graph ");
        printf("\n 2. BFS Traversal");


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

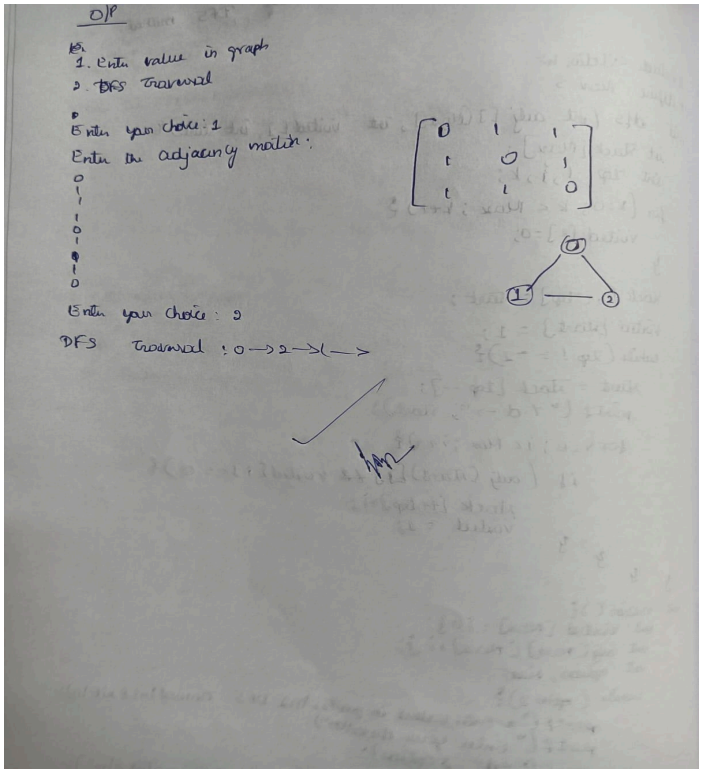
```
        {
            case 1: printf("\n Enter the adjacency matrix:\n");
                    for(i=0;i<MAX;i++){
                        for(j=0;j<MAX;j++){
                            scanf("%d",&adj[i][j]);
                        }
                    }
                    break;

            case 2: printf("BFS Traversal:");
                    bfs(adj,visited,0);
                    break;

}
}while(option!=3);
return 0;
}
```

```
 ********Main Menu***********

 1. Enter values in graph
 2. BFS Traversal

 Enter your choice:1

 Enter the adjacency matrix:
0
1
1
1
1
0
1
1
1
1
0
1
1
1
1
0

 ********Main Menu***********

 1. Enter values in graph
 2. BFS Traversal

 Enter your choice:2
BFS Traversal:0->1->2->3->
```

9b) Write a program to check whether given graph is connected or not using DFS method.



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

#include <stdio.h>
#define Max 5
void dfs (int adj [] [Max], int visited [], int start) {
    int stack [Max];
    int top = -1, j, k;
    for (k=0; k < Max ; k++) {
        visited [k] = 0;
    }
    stack [++top] = start;
    visited [start] = 1;
    while (top != -1) {
        start = stack [top--];
        printf ("%d ->", start);
        for (j=0 ; j< Max ; j++) {
            if (adj [start] [j] && visited [j] == 0) {
                stack [++top] = j;
                visited = 1;
            }
        }
    }
}
int main () {
    int visited [Max] = {0};
    int adj [Max] [Max], i, j;
    int option, size;
    while (option != 3) {
        printf ("1. Enter values in graph. \n2. DFS traversal \n3. Exit \n");
        printf ("Enter your choice \n");
        scanf ("%d", &option);
        switch (option) {
            case 1 : printf ("Enter the adjacency matrix :\n");
                     for (i=0 ; i < Max ; i++) {
                         for (j=0 ; j < Max ; j++) {
                             scanf ("%d", &adj [i] [j]);
                         }
                     }
                     break;
            case 2 : printf ("\n DFS traversal : ");
                     dfs (*adj, visited, 0);
                     printf ("\n");
                     break;
            case 3 : exit (0);
            default : printf ("Invalid choice ! ");
        }
    }
}
```



```
O/P

1. Enter value in graph
2. DFS traversal

Enter your choice : 1
Enter the adjacency matrix :
0
1
1
1
0
1
1
1
0

Enter your choice : 2
DFS traversal : 0 --> 2 --> 1 -->
```

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

void dfs(int adj[][MAX], int visited[], int start)
{
    int stack[MAX];
    int top = -1, i, k;


    for(k = 0; k < MAX; k++) {
        visited[k] = 0;
    }


    stack[++top] = start;
    visited[start] = 1;


    while(top != -1)
    {
        start = stack[top--];
        printf("%d -> ", start);


        for(i = 0; i < MAX; i++)
        {
            if(adj[start][i] && visited[i] == 0)
            {
                stack[++top] = i;
                visited[i] = 1;
            }
        }
    }
}

int main()
{
    int visited[MAX] = {0};
    int adj[MAX][MAX], i, j;
    int option, size;
```

```
    do {

        printf("\n 1. Enter values in graph ");
        printf("\n 2. DFS Traversal");
        printf("\n 3. Exit");
        printf("\n \n Enter your choice: ");
        scanf("%d", &option);

        switch(option)
        {
            case 1:
                printf("\n Enter the adjacency matrix:\n");
                for(i = 0; i < MAX; i++) {
                    for(j = 0; j < MAX; j++) {
                        scanf("%d", &adj[i][j]);
                    }
                }
                break;

            case 2:
                printf("\n DFS Traversal: ");
                dfs(adj, visited, 0);
                printf("\n");
                break;

            case 3:
                printf("\n Exiting program.\n");
                break;

            default:
                printf("\n Invalid option. Please try again.\n");
        }
    } while(option != 3);

    return 0;
}
```

```
 1. Enter values in graph
 2. DFS Traversal
 3. Exit

 Enter your choice: 1

 Enter the adjacency matrix:
0
1
1
1
0
1
1
1
0

 1. Enter values in graph
 2. DFS Traversal
 3. Exit

 Enter your choice: 2

 DFS Traversal: 0 -> 2 -> 1 ->

 1. Enter values in graph
 2. DFS Traversal
 3. Exit

 Enter your choice: 3

 Exiting program.
```