VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SANTHOSH N (1BM23CS302)

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 **SANTHOSH N(1BM23CS302),** who is bonafide student of **B.M.S.College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering inComputer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

Lab faculty Incharge Name Assistant Professor Department of CSE, BMSCE	Geetha N Professor Department of CSE, BMSCE

Index

Github Link: https://github.com/SANTHoshN302/1BM23CS302-SANTHOSH_N-.git

Program 1

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

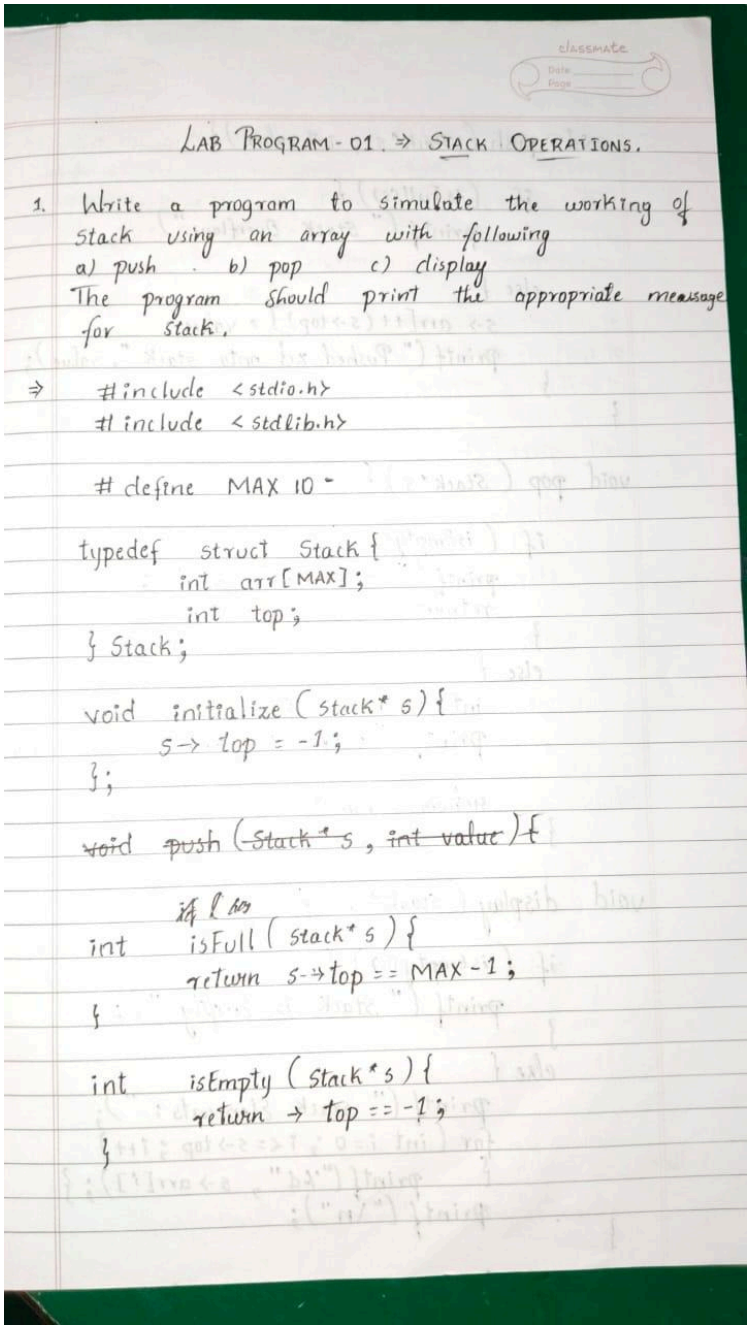
a) Push

b) Pop

c) Display

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

SOFT COPY:



LAB PROGRAM - 01 ⇒ STACK OPERATIONS.

1. Write a program to simulate the working of stack using an array with following
a) push b) pop c) display
The program should print the appropriate meaisage for stack.

⇒

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

#define MAX 10

typedef struct Stack {
        int arr[MAX];
        int top;
} Stack;

void initialize (Stack* s){
     s-> top = -1;
};

void push (Stack* s, int value){

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

int  isEmpty (Stack* s){
     return -> top == -1;
}
```

```
void push (Stack *s, int value) {
    if (isFull(s)) {
        printf ("Stack Overflow")
    else {
        s-> arr[++(s->top)] = value;
        printf ("Pushed %d onto stack", value);
    }
}

void pop (Stack *s) {
    if (isEmpty(s)) {
        printf ("Stack Underflow");
        return -1;
    }
    else {
        int value = s-> arr[(s->top)--];
        printf ("popped %d from stack", value);

        return value;
    }

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

```
int main()
{
    stack s;
    initialize(&s);

    push(&s, 10);
    push(&s, 35);
    display(&s);

    pop(&s);
    display(&s);

    return 0;
}
```

Output:

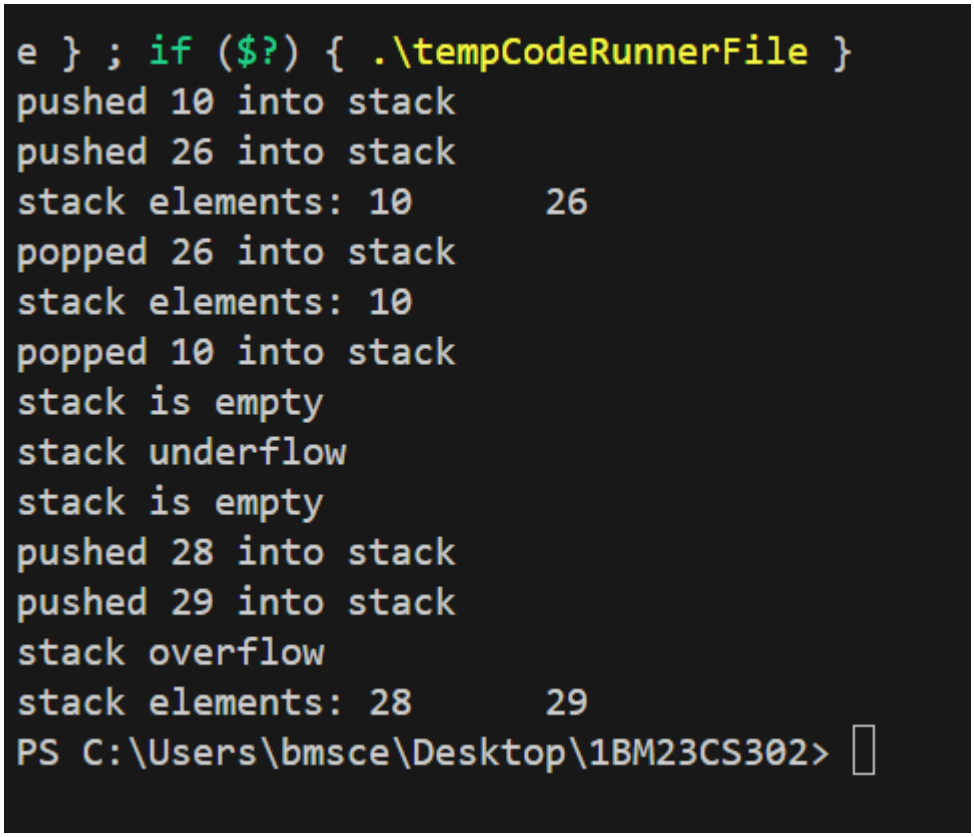
```
Pushed 10 onto Stack
Pushed 35 onto Stack
Stack Elements : 10 35
Popped 35 from Stack
Stack Elements : 10
```

Code:

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 2
typedef struct stack{
    int arr[MAX];
    int top;
}stack;

void initialize(stack* s){
    s-> top=-1;
}
int isFull(stack* s){
    return s-> top == MAX-1;
}
int isEmpty(stack* s){
    return s-> top == -1;
}
void push(stack* s, int value){
    if (isFull(s)){
        printf("stack overflow\n");
    }
    else{
        s->arr[++(s->top)]=value;
        printf("pushed %d into stack\n",value);
    }
}
int pop(stack* s){
    if (isEmpty(s)){
        printf("stack underflow\n");
        return -1;
    }
    else{
        int value = s->arr[(s->top)--];
        printf("popped %d into stack\n",value);
        return value;

    }
}
void display(stack* s){
    if (isEmpty(s)){
```

```
        printf("stack is empty\n");
    }
    else{
        printf("stack elements: ");
        for(int i=0; i<=s->top;i++){
            printf("%d\t",s->arr[i]);
        }
        printf("\n");
    }
}
int main(){
    stack s;
    initialize(&s);

    push(&s,10);
    push(&s,26);
    display(&s);

    pop(&s);
    display(&s);
    pop(&s);
    display(&s);
    pop(&s);
    display(&s);

    push(&s,28);
    push(&s,29);
    push(&s,30);
    display(&s);

    return 0;

}
```

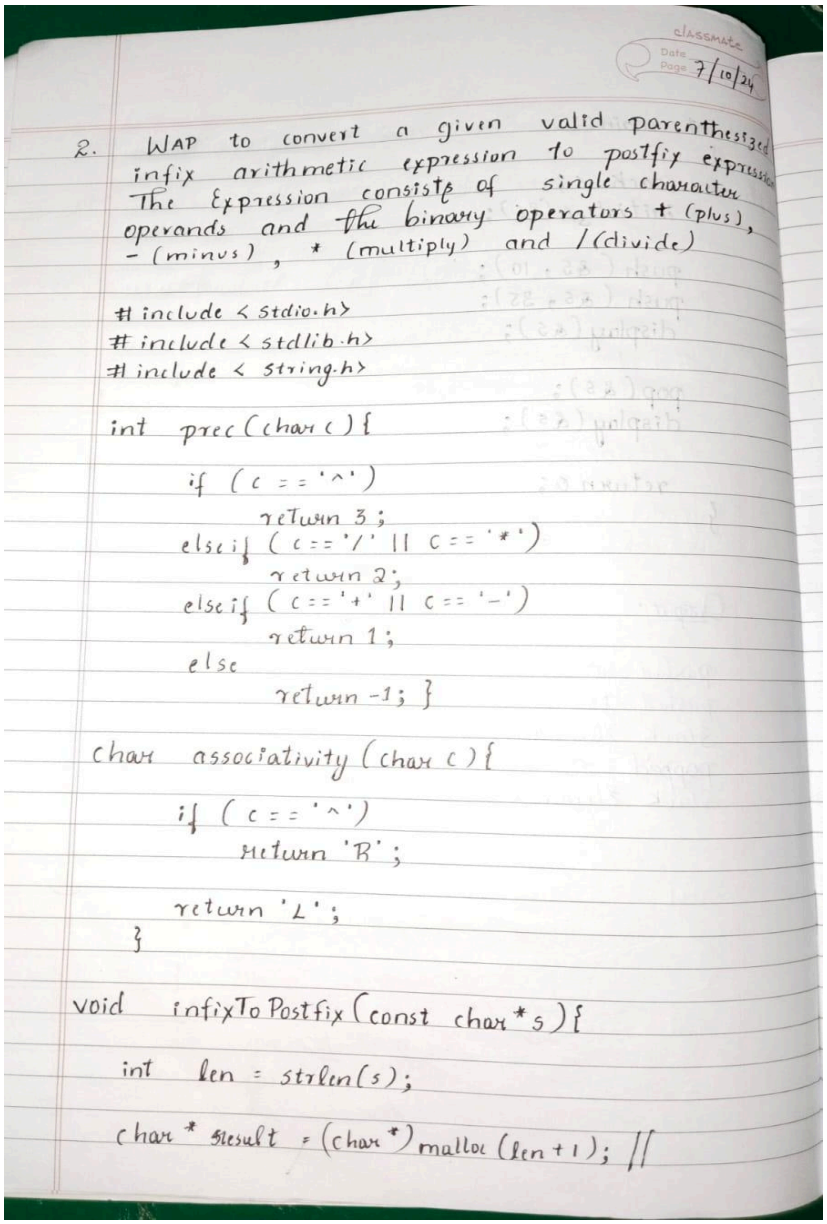
OUTPUT:


e } ; if ($?) { .\tempCodeRunnerFile }
pushed 10 into stack
pushed 26 into stack
stack elements: 10 26
popped 26 into stack
stack elements: 10
popped 10 into stack
stack is empty
stack underflow
stack is empty
pushed 28 into stack
pushed 29 into stack
stack overflow
stack elements: 28 29
PS C:\Users\bmsce\Desktop\1BM23CS302>

Program 2:

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

SOFT COPY



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

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

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

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

void infixToPostfix(const char *s){
    int len = strlen(s);
    char * result = (char *) malloc (len + 1); //
```

```
char* stack = (char*)malloc(len);

int resultIndex = 0;
int stackIndex = -1;

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

for(int i=0; i<len; i++){
    char c = s[i];

if((c>='a' && c<='z') || (c>='A' && c<='Z')
    || (c>='0' && c<='9')){
    result[resultIndex++] = c;
}
else if (c=='('){
    stack[++stackIndex] = c;
}
else if (c==')'){
    while(stackIndex>=0 && stack[stackIndex]
        !='('){
        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);
free(result);
free(stack);

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

O/p: ~~abcd^e-fgh*+^*+i-~~
abcd^e-fgh*+^*+i-

Seen
GL
07/24

CODE :

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

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

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

    return 'L';
}
void infixToPostfix(const char*s){
    int len=strlen(s);
    char result[100];
    char  stack[100];

    int resultIndex=0;
    int stackIndex=-1;
    for (int i=0;i<len;i++){
        char c=s[i];

        if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c>='0'&&c<='9')){
            result[resultIndex++]=c;
        }
        else if(c=='('){
            stack[++stackIndex]=c;
        }
```

```
        else if(c==')'){
            while(stackIndex>=0 && stack[stackIndex]!='('){
                result[resultIndex++]=stack[stackIndex--];
            }
            stackIndex--;
        }
        else{
            while(((stackIndex>=0) &&
(prec(c)<prec(stack[stackIndex]))||(prec(c)==prec(stack[stackIndex])))&&associativi
ty(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[100];
    printf("enter any expression\n");
    gets(exp);
    infixToPostfix(exp);
    return 0;
}
```

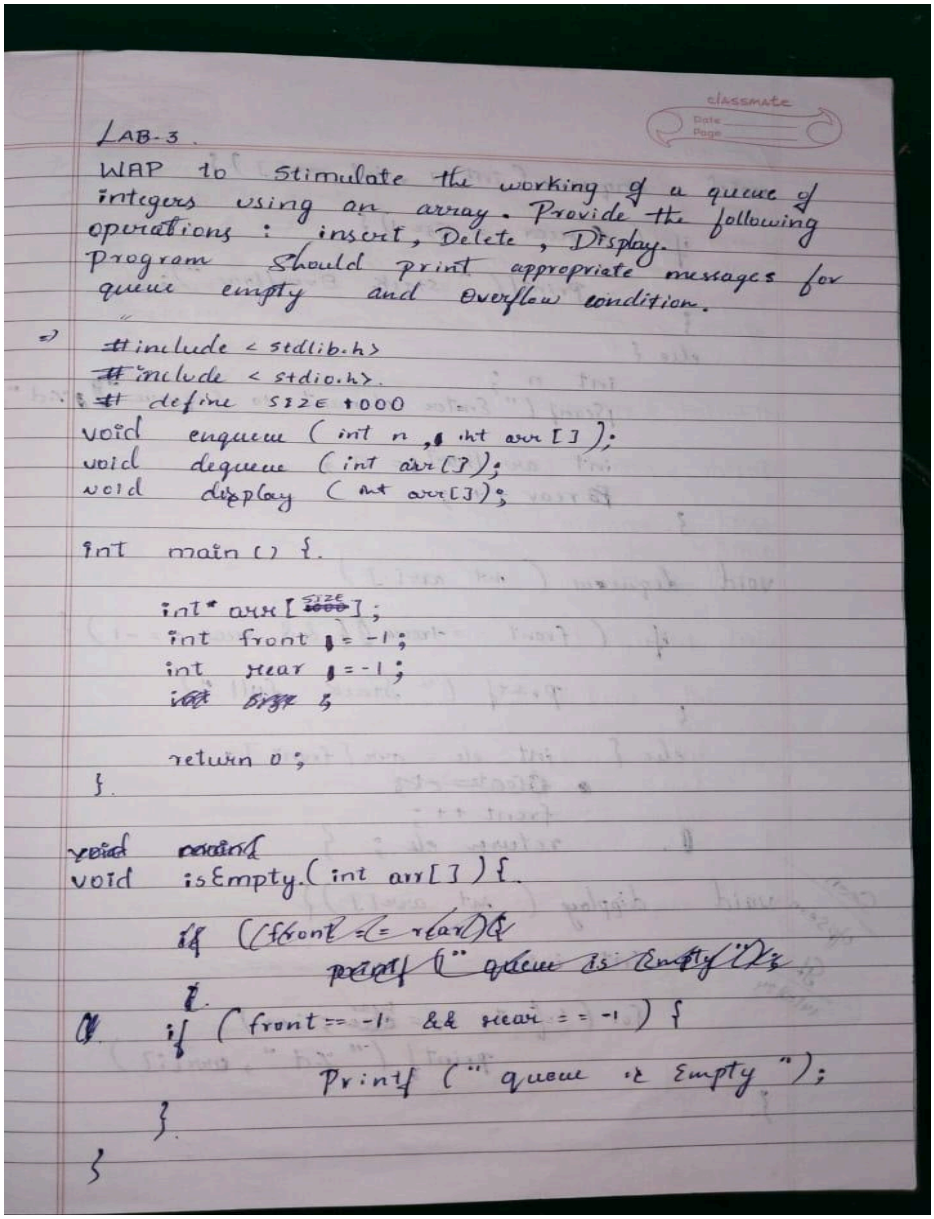
OUTPUT:

```
enter any expression
a+b*(c^d-e)^(f+g*h)-i
abcd^e-fgh*+^*+i-
PS C:\Users\bmsce\Desktop\1BM23CS302>
```

PROGRAM 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.

SOFT COPY



LAB-3.

WAP to Stimulate the working of a queue of integers using an array. Provide the following operations : insert, Delete, Display. Program should print appropriate messages for queue empty and Overflow condition.

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 1000
void enqueue (int n, int arr[]);
void dequeue (int arr[]);
void display (int arr[]);

int main () {

    int* arr[SIZE];
    int front = -1;
    int rear = -1;

    return 0;
}

void isEmpty(int arr[]){
    if ((front == rear)&&
        printf("queue is Empty");
    }
    if (front == -1 && rear == -1) {
        Printf ("queue is Empty");
    }
}
```

```
void enqueue (int n, int arr[]) {
    if ( rear == size-1) {
        printf (" Stack Overflow ");
    }
    else {
        int n ;
        scanf (" Enter element to Enqueue : %d ", &n);
        int arr [rear] = n ;
        rear ++ ;
    }

void dequeue ( int arr[] )
    if ( front == -1 && rear == -1 ) {
        printf (" Stack full ")
    }
    else { int ele = arr [front] ;
        front ++ ;
        return ele ; }

void display ( int arr[] ) {
    int i ;
    for ( i=front; i<= rear ; i++)
        printf (" %d ", arr[i] )
}
```

14/10/24

Output :

⇒ enqueued : 5
enqueued : 6
enqueued : 7
Queue is overflow.
Dequeued : 5
Dequeued : 6
Dequeued : 7
Queue is underflow.

CODE :

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10000

void enqueue(int n, int arr[], int *rear);
int dequeue(int arr[], int *front, int *rear);
void display(int arr[], int front, int rear);

int main() {
    int arr[SIZE];
    int front = 0;
    int rear = 0;


    enqueue(5, arr, &rear);
    enqueue(10, arr, &rear);
    display(arr, front, rear);
    printf("Dequeued: %d\n", dequeue(arr, &front, &rear));
    display(arr, front, rear);

    return 0;
}

void enqueue(int n, int arr[],int *rear) {
    if (*rear == SIZE) {
        printf("Queue is full\n");
    } else {
        printf("Enter element to enqueue: ");
        scanf("%d", &n);
        arr[*rear] = n;
        (*rear)++;
    }
}

int dequeue(int arr[], int *front, int *rear) {
    if (*front == *rear) {
        printf("Queue is empty\n");
        return -1;
    } else {
        int m = arr[*front];
        (*front)++;
        return m;
```

```
    }
}

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

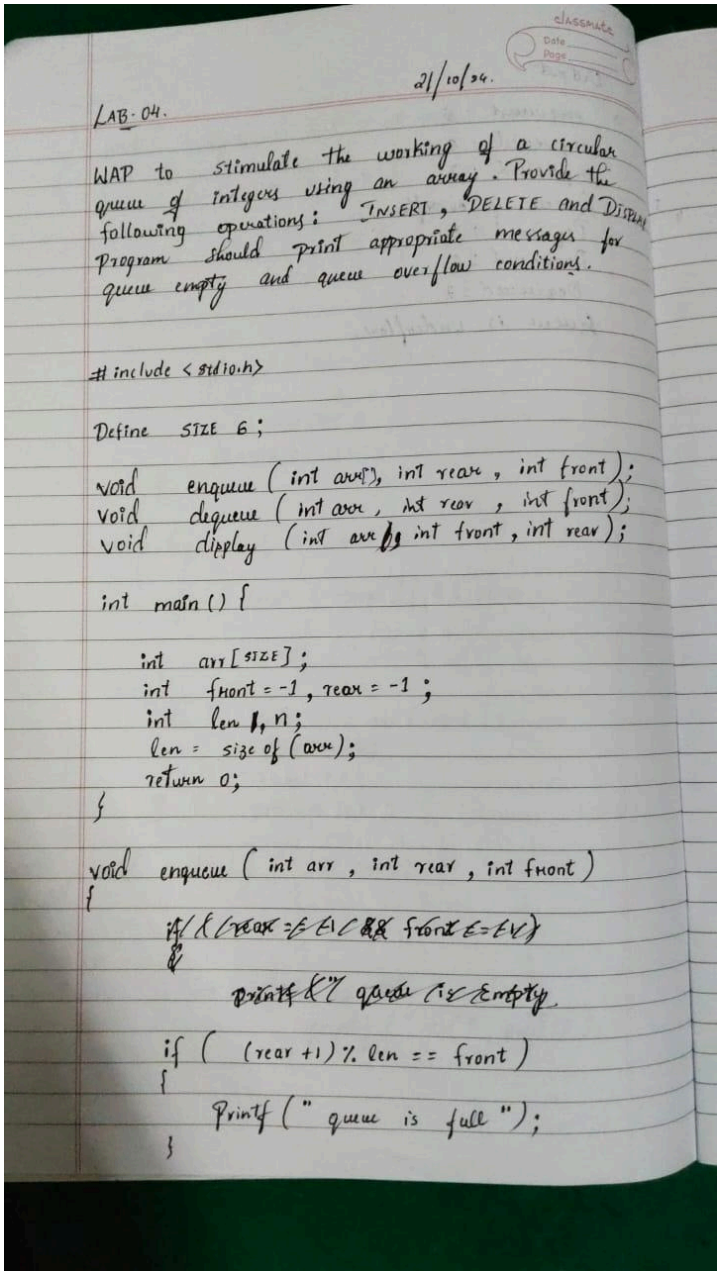
OUTPUT :

```
e } ; if ($?) { .\tempCodeRunnerFile }
Enter element to enqueue: 5
Enter element to enqueue: 6
Enter element to enqueue: 2
Queue is full
Queue elements: 5 6 2
Dequeued: 5
Dequeued: 6
Dequeued: 2
Queue is empty
PS C:\Users\bmsce\Desktop\1BM23CS302>
```

PROGRAM 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

SOFT COPY



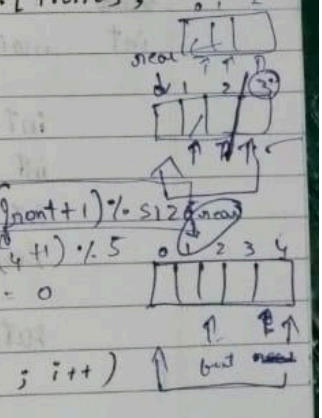
21/10/24.

LAB-04.

WAP to stimulate the working of a circular queue of integers using an array. Provide the following operations: INSERT, DELETE and Display. Program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>

Define SIZE 6;

void enqueue (int arr[], int rear, int front);
void dequeue (int arr, int rear, int front);
void display (int arr[], int front, int rear);

int main() {

    int arr[SIZE];
    int front = -1, rear = -1;
    int len, n;
    len = size of (arr);
    return 0;
}

void enqueue (int arr, int rear, int front)
{
    if ((rear == -1 && front == -1)
    {
        printf("queue is empty
    if ( (rear +1) % len == front )
    {
        Printf (" queue is full ");
    }
```

```
    else {
            printf (" Enter Element to Enqueue : ");
            scanf (" %d \n ", &n);

            arr[rear] = n ;
            rear ++ ;
        }
}.

void dequeue ( int arr , int rear , int front )
{
    if ( front == -1 && rear == -1 )
    {
        printf (" queue is empty ");
    }

    else {
            int element = arr[front];
            front ++ ;

            return element ;
        }
}.

void display ( int arr )
{
    for ( int i=fro ; i<= SIZE ; i++ )
    {
        printf (" %d ", arr[i]);
    }
}
```



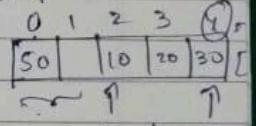
```
void display ( int arr , int rear ; int front )
{       front == 0 ; rear == SIZE - 1 ;
        for ( int i = front ; i <= rear ; i++ )
                printf ( " %d " , arr[i] );
}
```

Modify and rewrite the code.

⇒

```
#include <stdio.h>

Define SIZE 5;

void enqueue ( int arr[] , int front , int rear );
Void dequeue ( int arr[] , int front , int rear );
void display ( int arr[] , int front , int rear );

void
int main () {

        int arr[SIZE];
        int front == -1 , rear == -1 ;
        int len , n ;
        len = size of (arr);

        return 0;
}
```

```
void enqueue ( int arr[], int front, int rear)
{
    if ( (rear + 1) % len == front )
    {
        printf (" queue is full ") ✓
    }

    else { front == -1 ; rear == -1;
                                front = rear = 0
        printf (" enter element : ");
        scanf (" %d ", &n);
        arr[rear] = n;
        rear ++ ;
        rear = (rear+1) % SIZE
    }
}
```



(4 + 1) % 5 rear = SIZE - 1
5 % 5 == 0 5 - 1 = 4

arr[rear] = n

```
void dequeue ( int arr[], int front, int rear)
{
    if ( rear == -1 && front == -1 )
    {
        printf (" queue is empty ");
    }
    else {
        int ele = arr[front];
        front = (front + 1) % len;
        return ele;
    }
}
```

```
Void display (int arr[], int rear, int front)
{
    front == 0 ; rear == SIZE - 1 ;   ⇒ overflow
    for( int i = front ; i <= rear ; i = (front + 1) % len)
    {
        Printf (" %d " , arr [i]);
    }
}
```

CODE :

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

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


void initializeQueue(Queue *q);
int isFull(Queue *q);
int isEmpty(Queue *q);
void insert(Queue *q, int value);
int delete(Queue *q);
void display(Queue *q);

int main() {
    Queue q;
    initializeQueue(&q);
    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(&q, value);
                break;
            case 2:
                value = delete(&q);
                if (value != -1) {
                    printf("Deleted value: %d\n", value);
```

```
                }
                break;
            case 3:
                display(&q);
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}

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

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

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


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

```c
int delete(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Queue is empty.\n");
        return -1;
    }
    return q->items[q->front++];
}


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:

Output

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 23
Inserted 23 into the queue.

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

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 23 25

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted value: 23
```

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 56
Inserted 56 into the queue.

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 78
Queue Overflow! Cannot insert.
```

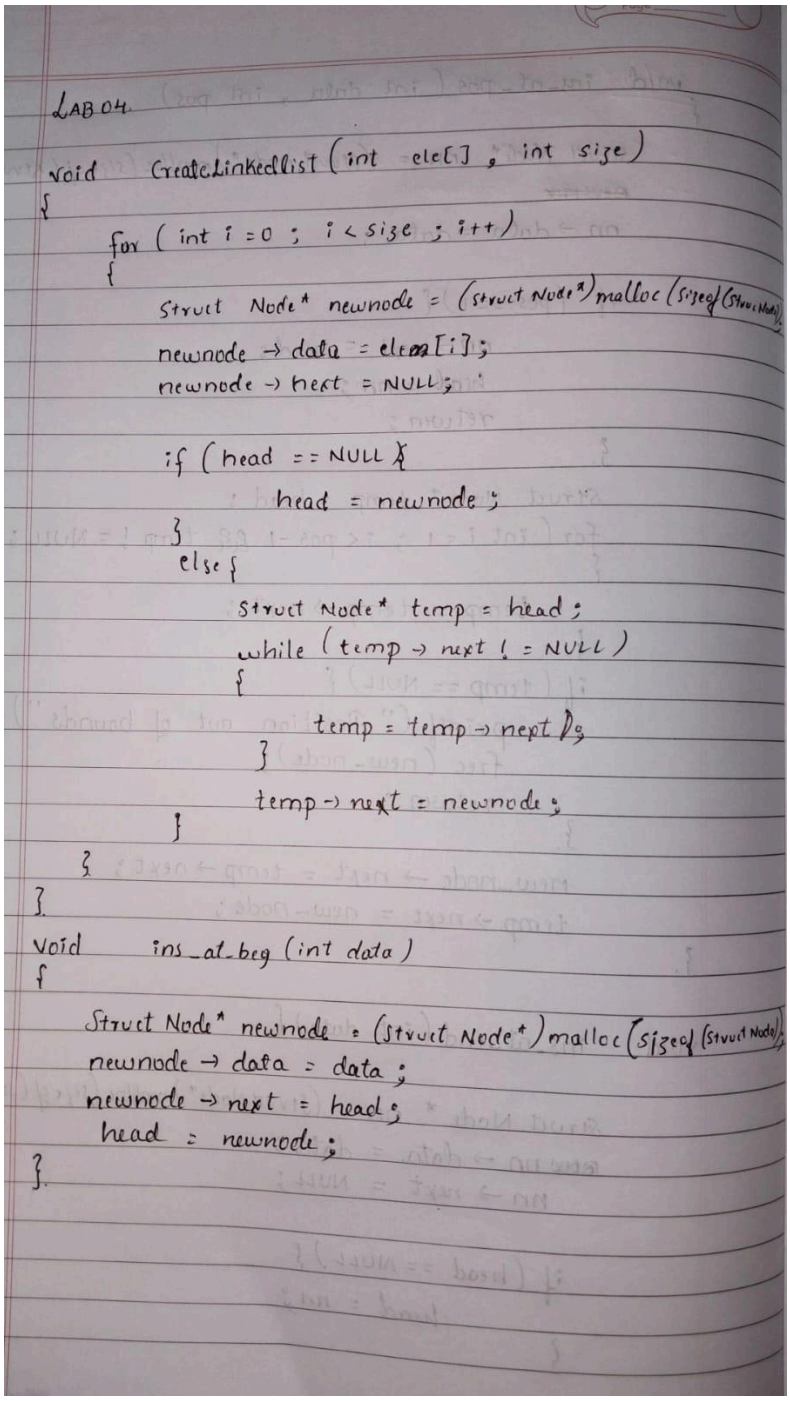
Output

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! Queue is empty.

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

PROGRAM 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.
c) Display the contents of the linked list.

SOFT COPY



```
LAB 04

void CreateLinkedlist (int ele[], int size)
{
    for (int i=0; i<size; i++)
    {
        Struct Node* newnode = (struct Node*)malloc(sizeof(StructNode));
        newnode -> data = elem[i];
        newnode -> next = NULL;

        if (head == NULL){
            head = newnode;
        }
        else{
            Struct Node* temp = head;
            while (temp -> next != NULL)
            {
                temp = temp -> next;
            }
            temp -> next = newnode;
        }
    }
}
void ins_at_beg (int data)
{
    Struct Node* newnode = (Struct Node*)malloc(sizeof(Struct Node));
    newnode -> data = data;
    newnode -> next = head;
    head = newnode;
}
```

```
void ins_at_pos ( int data , int pos)

    Struct Node* nn = (struct node*) malloc (sizeof (struct Node));
    ~~newnode~~
    nn → data = data ;

    if (pos == 1) {
            nn → next = head ;
            head = nn ;
            return ;
    }.
    Struct Node* temp = head ;
    for ( int i = 1 ; i < pos - 1 && temp != NULL ; i++)
    {
        temp = temp → next ;
    }.
    if ( temp == NULL) {
        printf (" Position out of bounds ");
        free (new_node);
        return ;
    }.
    new_node → next = temp → next ;
    temp -> next = new_node ;
}.

void ins_at_end (int data) {

    Struct Node * nn = (Struct Node*) malloc (sizeof (Struct No
    nn → data = data ;
    nn → next = NULL ;

    if (head == NULL) {
            head = nn ;
    }
```

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

void display() {

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

Output

```
1. Insert at Begginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit
Enter your choice : 1
Enter data : 21

1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit.
Enter your choice : 2
Enter data : 20
Enter position : 1

1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit
Enter your choice : 3
Enter data : 22

1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit
Enter your choice : 4
20 21 22
```

Seen
16/12/24

CODE :

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


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


struct Node* head = NULL;


void createLinkedList(int elements[], int size) {
    for (int i = 0; i < size; i++) {
        struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = elements[i];
        new_node->next = NULL;

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


void insertAtBeginning(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = head;
    head = new_node;
}


void insertAtPosition(int data, int position) {
```

```
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;

    if (position == 1) {
        new_node->next = head;
        head = new_node;
        return;
    }

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

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

    new_node->next = temp->next;
    temp->next = new_node;
}


void insertAtEnd(int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;

    if (head == NULL) {
        head = new_node;
        return;
    }

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

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

int main() {
    int choice, data, position, size;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create Linked List\n");
        printf("2. Insert at Beginning\n");
        printf("3. Insert at Position\n");
        printf("4. Insert at End\n");
        printf("5. Display Linked List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of elements: ");
                scanf("%d", &size);
                int elements[size];
                printf("Enter the elements: ");
                for (int i = 0; i < size; i++) {
                    scanf("%d", &elements[i]);
                }
                createLinkedList(elements, size);
                break;

            case 2:
                printf("Enter the data to insert: ");
                scanf("%d", &data);
                insertAtBeginning(data);
```

```
                break;

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

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

            case 5:
                displayLinkedList();
                break;

            case 6:
                printf("Exiting...\n");
                exit(0);

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

    return 0;
}
```

OUTPUT

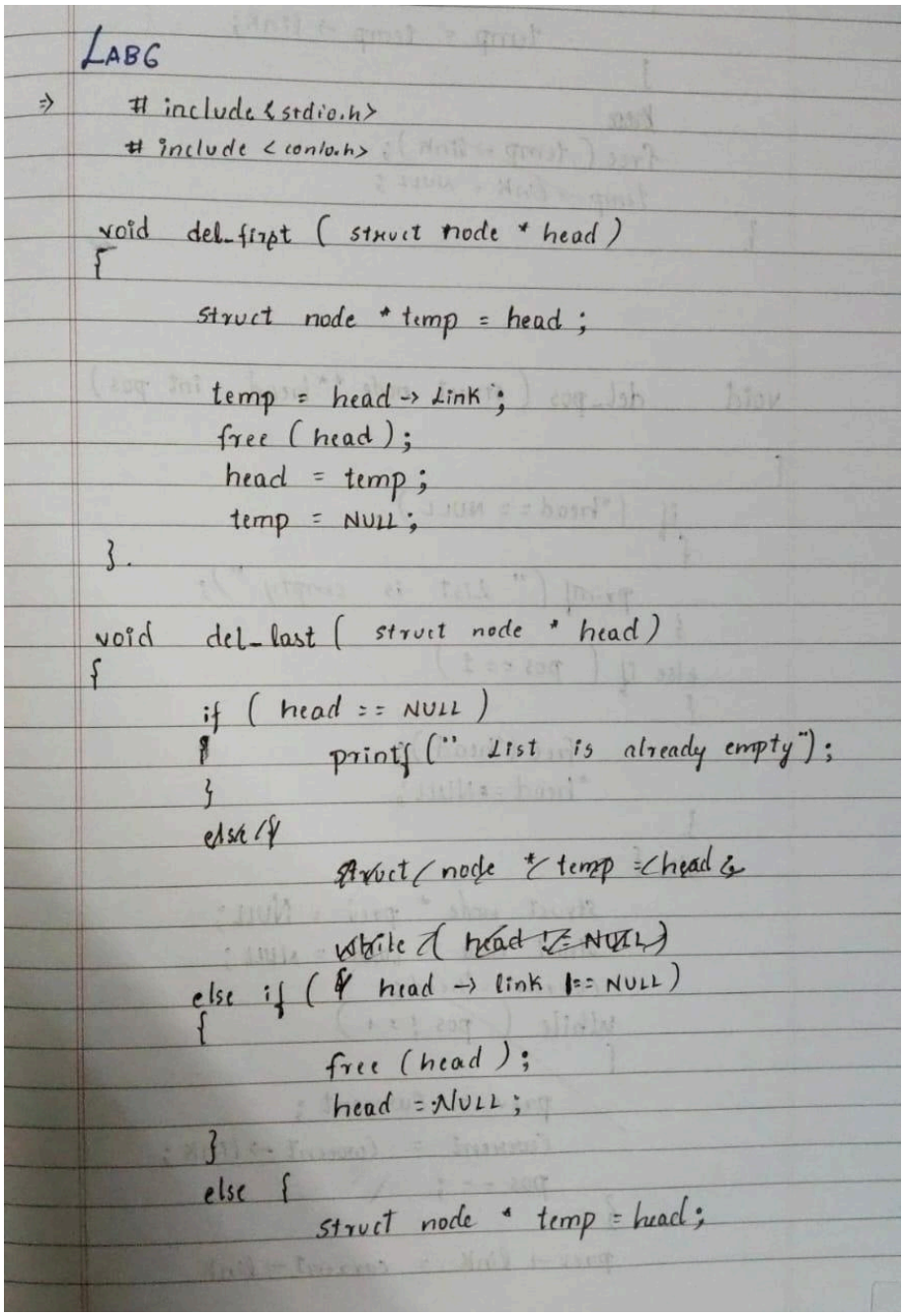
```
Menu:
1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display Linked List
6. Exit
Enter your choice: 1
Enter the number of elements: 2
Enter the elements: 1 3

Menu:
1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display Linked List
6. Exit
Enter your choice: 2
Enter the data to insert at the beginning: 0

Menu:
1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display Linked List
6. Exit
Enter your choice: 2
Enter the data to insert at the beginning: 0
```

```
Menu:
1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display Linked List
6. Exit
Enter your choice: 3
Enter the data to insert: 3
Enter the position: 3

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

Menu:
1. Create Linked List
2. Insert at Beginning
3. Insert at Position
4. Insert at End
5. Display Linked List
6. Exit
Enter your choice: 5
Linked List: 0 -> 0 -> 3 -> 1 -> 3 -> 4 -> NULL
```

PROGRAM 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

SOFT COPY :



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

void del_first ( struct node * head )
{
        struct node * temp = head ;

        temp = head -> Link ;
        free ( head );
        head = temp ;
        temp = NULL ;
}.

void del_last ( struct node * head )
{
        if ( head == NULL )
        {
                printf (" List is already empty");
        }
        else if
                struct node * temp = head ;
                while ( head != NULL )
        else if ( head -> link != NULL )
        {
                free ( head );
                head = NULL ;
        }
        else {
                struct node * temp = head ;
```

```
while ( temp -> link -> link != NULL )
{
        temp = temp -> link;
}
free ( temp -> link );
temp -> link = NULL;
}.

void    del_pos ( struct node **head, int pos)
{
    if (*head == NULL)
    {
        printf (" List is empty ");
    }
    else if ( pos == 1 )
    {
        free (*head);
        *head = NULL;
    }.
    else {
        struct node * prev = NULL;
        struct node * curr = NULL;
        current = *head;
        while ( pos != 1 )
        {
            prev = current;
            current = current -> link;
            pos --;
        }
        prev -> link = current -> link
```

```
    free (current);
    Curr = NULL;
}
```

Seen
o/p seen.

CODE :

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

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

void insert_to_end(struct node* head,int data){
    struct node * ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = data;
    temp->link = NULL;
    while(ptr->link != NULL){
        ptr = ptr->link;
    }
   ptr->link = temp;
}

struct node* del_first(struct node* head){
    struct node* curr = head;
    if(head == NULL){
        printf("list id empty");
    }
    else{
            head = curr->link;
            free(curr);
            curr = NULL;
        }
    return head;
}

struct node* del_last(struct node* head){

    if(head == NULL){
        printf("list is already empty");
    }
    else if(head->link == NULL){
        free(head);
```

```
        head = NULL;
    }
    else{
        struct node* curr = head;
        while(curr->link->link != NULL){
            curr = curr->link;
        }
        free(curr->link);
        curr->link = NULL;
    }

    return head;
}

struct node* del_pos(struct node**head, int pos){

    struct node * prev = *head;
    struct node * curr = *head;

    if(head == NULL){
        printf("list is already empty");
    }
    else if(pos == 1){
        *head = curr->link;
        free(curr);
        curr = NULL;
    }
    else{
        while(pos != 1){
            prev = curr;
            curr = curr->link;
            pos--;
        }
        prev->link = curr->link;
        free(curr);
        curr = NULL;
    }
    return *head;
}

void display_data(struct node* head){
```

```
    struct node* ptr = head;
    while(ptr != NULL){
        printf("%d\t",ptr->data);
        ptr = ptr->link;
    }
}

int main(){
    int num;
    struct node* head = malloc(sizeof(struct node));

    head->data = 45;
    head->link = NULL;

    while(1){
    printf("\n enter your choice:\n1.insert to list\n2.delete first
node\n3.delete last node\n4.delete at certain position\n5.exit\n");
    scanf("%d",&num);

    switch(num){

        case 1:
                int a;
                printf("enter data to insert: ");
                scanf("%d",&a);
                insert_to_end(head,a);

                break;
        case 2:
                head = del_first(head);
                break;
        case 3:
                head = del_last(head);
                break;
        case 4:
                int b;
                printf("enter the position of the data to delete: ");
                scanf("%d",&b);
                head = del_pos(&head,b);
                break;
```

```
        case 5: printf("program exceuted sucessfully");
                exit(0);
                break;
        default:
                printf("invalid input");
         }

         display_data(head);
    }

}
```

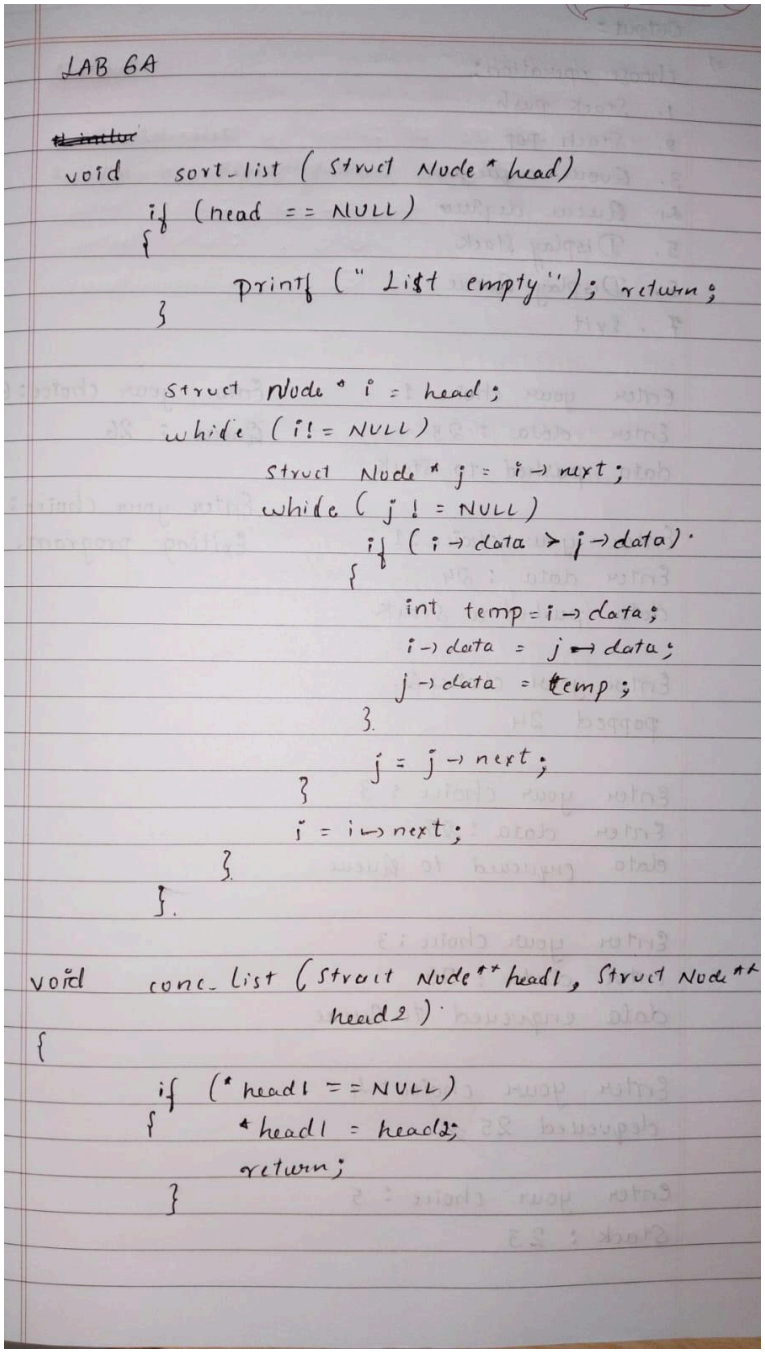
OUTPUT :

```
 enter your choice:
1.insert to list
2.delete first node
3.delete last node
4.delete at certain position
5.exit
1
enter data to insert: 67
45  23  90  67
 enter your choice:
1.insert to list
2.delete first node
3.delete last node
4.delete at certain position
5.exit
2
23  90  67
 enter your choice:
1.insert to list
2.delete first node
3.delete last node
4.delete at certain position
5.exit
3
23  90
```

PROGRAM 6A

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

SOFT COPY :



```
LAB 6A

void sort_list ( struct Node * head)
    if (head == NULL)
    {
        printf (" List empty "); return ;
    }

    struct Node * i = head ;
    while ( i != NULL )
        struct Node * j = i -> next ;
        while ( j != NULL )
            if ( i -> data > j -> data).
            {
                int temp = i -> data ;
                i -> data = j -> data ;
                j -> data = temp ;
            }.
            j = j -> next ;
        }
        i = i -> next ;
    }
}.

void conc_list (struct Node ** head1, struct Node **
                head2 ).
{
    if (* head1 == NULL)
    {   * head1 = head2;
        return ;
    }
```

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

void rev_list (struct Node **head)
{
    struct Node * prev = NULL;
    struct Node * curr = *head;
    struct Node * next = NULL;

    while (current != NULL)
    {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = ~~next~~ curr->next;
    }
    *head = prev;
}
```

Output

```
Enter Element of list1: 10 30 20
Enter Element of list2: 3 6 5
Sorted list1: 10 20 30
Sorted list2: 3 5 6
Concatenated list: 3 5 6 10 20 30
```

CODE :

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


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


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

void insert_end(struct Node** head, int data) {
    struct Node* new_node = create_node(data);
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new_node;
}

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

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

void sort_list(struct Node* head) {
    if (head == NULL) return;
    struct Node* i = head;
    struct Node* j = NULL;
    int temp;
    while (i != NULL) {
        j = i->next;
        while (j != NULL) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
            j = j->next;
        }
        i = i->next;
    }
}

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


void concatenate_lists(struct Node** head1, struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
```

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

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

    insert_end(&list1, 4);
    insert_end(&list1, 2);
    insert_end(&list1, 3);
    insert_end(&list1, 1);


    printf("Original List 1: ");
    print_list(list1);


    reverse_list(&list1);
    printf("Reversed List 1: ");
    print_list(list1);

    sort_list(list1);
    printf("Sorted List 1: ");
    print_list(list1);


    insert_end(&list2, 6);
    insert_end(&list2, 7);
    insert_end(&list2, 8);


    printf("Original List 2: ");
    print_list(list2);

    concatenate_lists(&list1, list2);
    printf("Concatenated List: ");
    print_list(list1);
```

```
    return 0;

}
```

OUTPUT :

```
Output

Original List 1: 4   2   3   1
Reversed List 1: 1   3   2   4
Sorted List 1: 1     2   3   4
Original List 2: 6   7   8
Concatenated List: 1     2   3   4   6   7   8
```

PROGRAM 6B

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

SOFT COPY :

```
void push (Stack *s, int value){
    if (isFull(s)) {
        printf ("Stack Overflow")
    else {
        s-> arr[++(s->top)] = value;
        printf ("Pushed %d onto stack ", value);
    }
}

void pop (Stack *s) {
    if (isEmpty(s)) {
        printf ("Stack Underflow");
        return -1;
    }
    else {
        int value = s-> arr[(s->top)--];
        printf ("popped %d from Stack", value);
        return value;
    }

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

```
void enqueue (int n, int arr[]) {
    if ( rear == size-1) {
        printf(" Stack Overflow ");
    }
    else {
        int n;
        scanf(" Enter element to Enqueue : %d ",
        int arr[rear] = n;
        rear ++;
    }

void dequeue ( int arr[] )
    if ( front == -1 && rear == -1) {
        printf (" Stack full ")
    }
    else { int ele = arr[front];
        front ++;
        return ele; }

void display ( int arr[]) {
    int i;
    for ( i=front; i<= rear ; i++)
        printf (" %d ", arr[i])
}
```

14/10/24

Output :

=)

```
choose operation:
1. Stack push
2. Stack pop
3. Queue enqueue
4. Queue dequeue
5. Display stack
6. Display Queue
7. Exit.

Enter your choice 1
Enter data : 23
data pushed to stack

Enter your choice : 1
Enter data : 24
data pushed to stack

Enter your choice: 2.
popped 24

Enter your choice : 3
Enter data : 25
data enqueued to Queue

Enter your choice : 3
enter data : 26
data enqueued to Queue

Enter your choice : 4
dequeued 25

Enter your choice : 5
Stack : 23

Enter your choice: 6
Queue : 26

Enter your choice: 7
Exiting program.
```

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 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;
    }
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}


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


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


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

```c
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;


    int choice, data;


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


        switch (choice) {
            case 1:
                printf("Enter data to push: ");
                scanf("%d", &data);
                push(&stackTop, data);
                printf("Data is pushed to stack.\n");
                break;
            case 2:
                data = pop(&stackTop);
                if (data != -1) {
                    printf("Popped from stack: %d\n", data);
                }
                break;
            case 3:
                printf("Enter data to enqueue: ");
                scanf("%d", &data);
                enqueue(&queueFront, &queueRear, data);
                printf("Data is enqueued.\n");
                break;
            case 4:
                data = dequeue(&queueFront, &queueRear);
```

```
                if (data != -1) {
                    printf("Dequeued from queue: %d\n", data);
                }
                break;
            case 5:
                printf("Stack: ");
                printList(stackTop);
                break;
            case 6:
                printf("Queue: ");
                printList(queueFront);
                break;
            case 7:
                printf("Exiting the Program.....\n");
                exit(0);
            default:
                printf("Invalid choice, please try again.\n");
        }
    }


    return 0;
}
```

OUTPUT :

Output

```
Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push: 23
data is pushed to stack

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push: 24
data is pushed to stack
```

Output

```
data is enqueued

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 2
Popped from stack: 25

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 2
Popped from stack: 24
```

Output

```
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push: 26
data is pushed to stack

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 5
Stack: 26    25  23

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 26    26  28
```

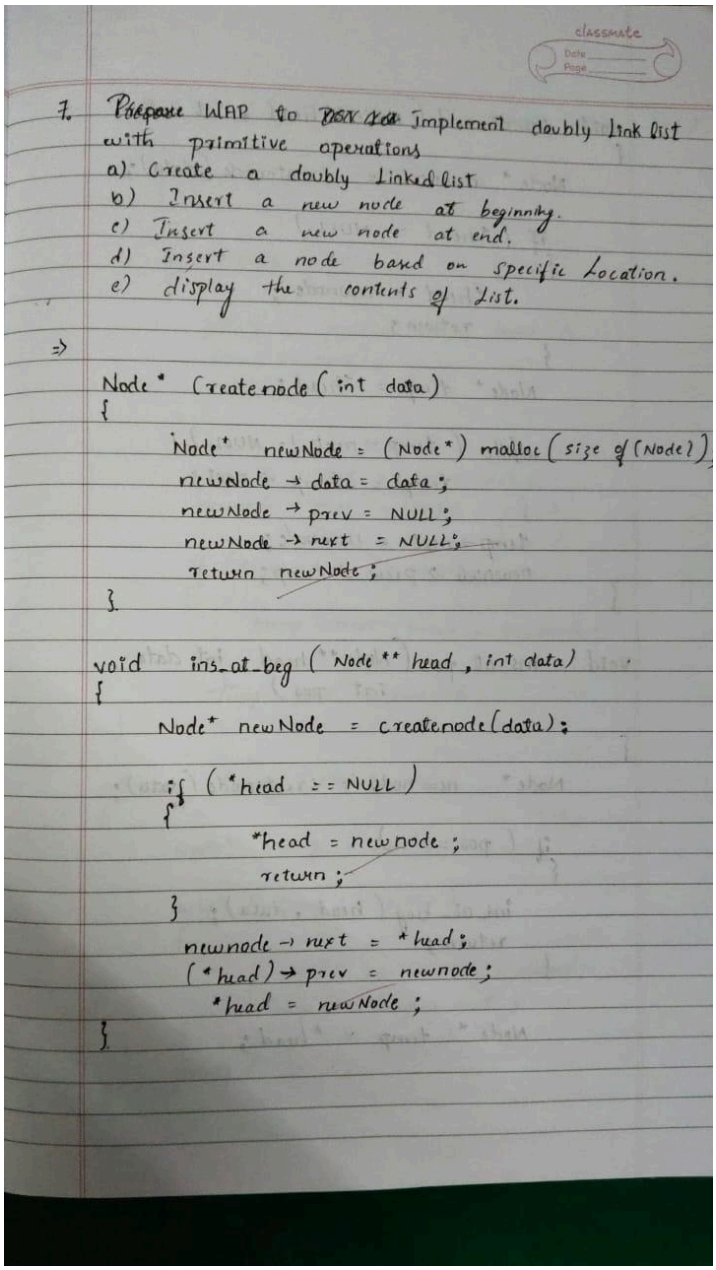
Output

```
data is pushed to stack

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 3
Enter data to enqueue: 26
data is enqueued

Choose operation:
1. Stack Push
2. Stack Pop
3. Queue Enqueue
4. Queue Dequeue
5. Display Stack
6. Display Queue
7. Exit
Enter your choice: 3
Enter data to enqueue: 26
data is enqueued
```

PROGRAM 7

WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

SOFT COPY :



7. Prepare WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list
b) Insert a new node at beginning.
c) Insert a new node at end.
d) Insert a node based on specific location.
e) display the contents of list.

=>

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

void ins_at_beg (Node** head, int data)
{
    Node* newNode = createnode(data);

    if (*head == NULL)
    {
        *head = newnode;
        return;
    }
    newnode -> next = *head;
    (*head) -> prev = newnode;
    *head = newNode;
}
```

```
void ins_at_end ( Node ** head , int data)
{
    Node * newnode = createnode (data);

    if ( *head == NULL )
    {
        *head = newnode ;
        return ;
    }
    Node * temp = *head ;

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

void ins_at_pos ( Node ** head , int data ,
                  int pos )

{
    Node * newnode = createnode (data);

    if ( pos == 1 )
    {
        ins_at_beg ( head , data);
        return ;
    }

    Node * temp = *head ;
```

```
for (int i = 1 ; i < pos - 1 && temp != NULL ; i++)
{
        temp = temp -> next;
}

if (temp == NULL)
{
    printf (" position out of bound ");
    free (newnode);
    return;
}

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

if (temp -> next != NULL)
{
        temp -> next -> prev = newnode;
}
    temp -> next = newnode;
}

void display (Node * head)
{
    Node * temp = head;

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

Output

1. Insert at Beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit

Enter your choice : 1
Enter data : 21

1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit.

Enter your choice : 2
Enter data : 20
Enter position : 1

1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit

Enter your choice : 3
Enter data : 22

Seen
16/12/24

1. Insert at beginning
2. Insert at position
3. Insert at end
4. Display list
5. Exit

Enter your choice : 4
20 21 22

CODE :

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

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}
void insertAtPosition(struct Node** head, int data, int position) {
    struct Node* newNode = createNode(data);
    if (position == 1) {
        insertAtBeginning(head, data);
        return;
    }

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

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

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

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

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

    while (1) {
        printf("\n1. Insert at Beginning\n");
        printf("2. Insert at 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 data: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                printf("data entered at beginning \n");
                break;
            case 2:
                printf("Enter data: ");
                scanf("%d", &data);
                printf("Enter position: ");
                scanf("%d", &position);
                insertAtPosition(&head, data, position);
                printf("data entered at position no %d\n",position);
                break;
            case 3:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                printf("data entered at end\n");
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;

}
```

OUTPUT :

```
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 1
Enter data: 22
data entered at beginning

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

1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 3
Enter data: 24
data entered at end

1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 3
Enter data: 21
data entered at end

1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 2
Enter data: 21
Enter position: 2
data entered at position no 2

1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 4
20 21 22 23 24 21
```

PROGRAM 8

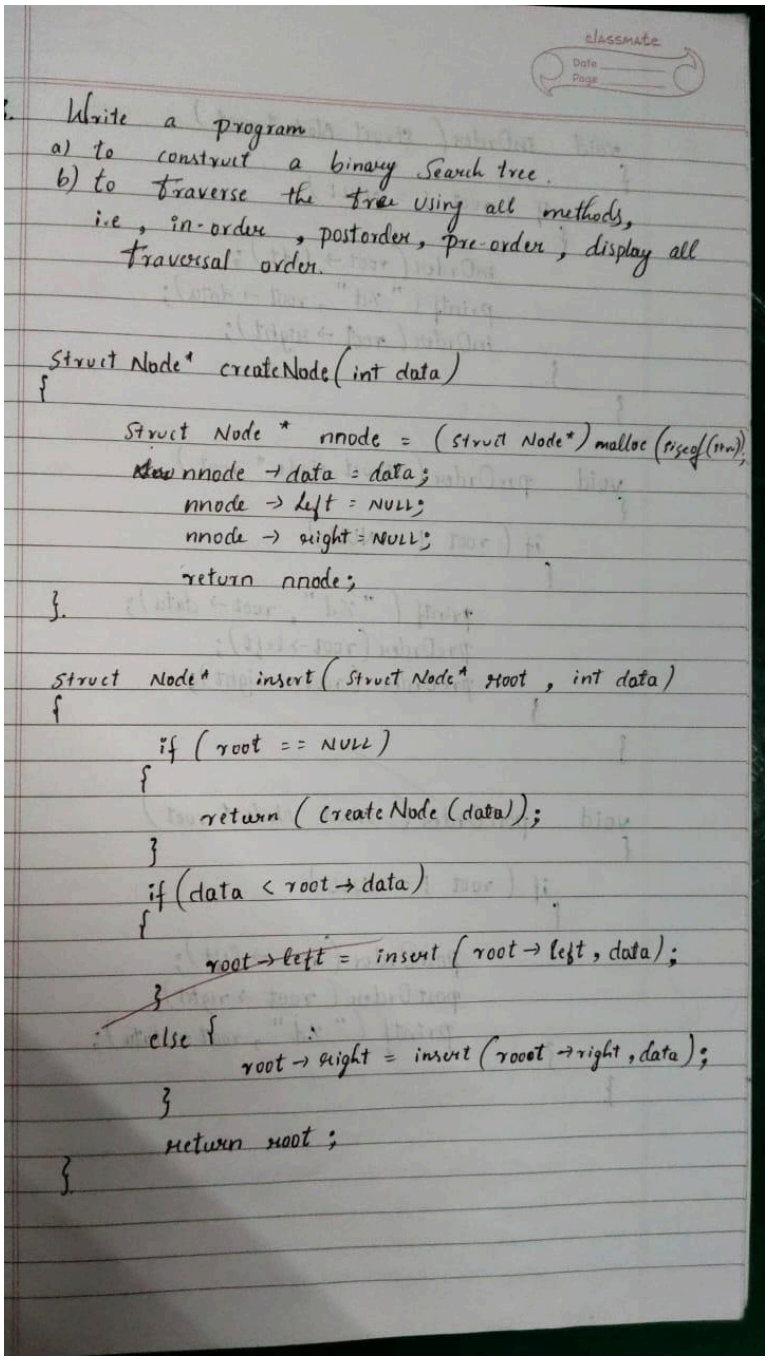
Write a program

a) ToconstructabinarySearchtree.

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

c) To display the elements in the tree.

SOFT COPY :



```
1. Write a program
a) to construct a binary Search tree.
b) to traverse the tree using all methods,
   i.e, in-order, postorder, pre-order, display all
   traversal order.

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

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

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

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

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

Q WAP to traverse a g

O/P

Enter no. of elements : 3

Enter Elements : 1 2 3

In-order traversal : 1 2 3
pre-order traversal : 1 2 3
post-order traversal : 3 2 1

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

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");
    while (1) {
        printf("\n1. Insert\n2. Inorder Traversal\n3. Preorder Traversal\n4.
Postorder Traversal\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preorderTraversal(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
```

```
                postorderTraversal(root);
                printf("\n");
                break;
            case 5:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT :

Output

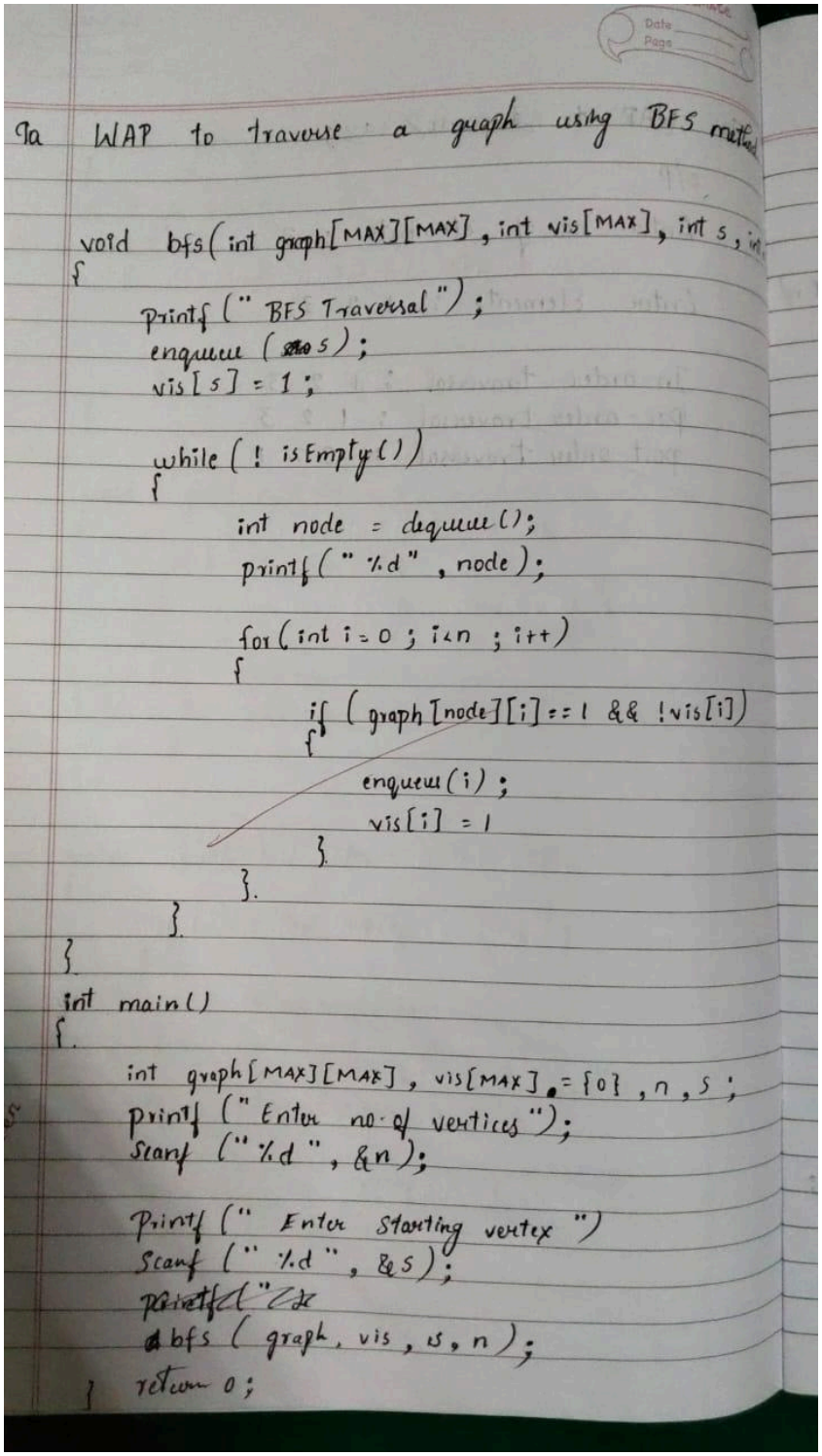
```
Binary Search Tree Traversals:
Inorder Traversal: 20 30 40 50 60 70 80
Preorder Traversal: 50 30 20 40 70 60 80
Postorder Traversal: 20 40 30 60 80 70 50


=== Code Execution Successful ===
```

PROGRAM 9

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

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

SOFT COPY



```
9a  WAP to traverse a graph using BFS meth

void bfs(int graph[MAX][MAX], int vis[MAX], int s, in
{
    Printf (" BFS Traversal ");
    enqueue (s);
    vis[s] = 1;

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

        for(int i=0 ; i<n ; i++)
        {
            if ( graph[node][i]==1 && !vis[i])
            {
                enqueue(i);
                vis[i] = 1
            }
        }
    }
}
int main()
{
    int graph[MAX][MAX], vis[MAX]={0}, n, s;
    printf ("Enter no. of vertices");
    Scanf ("%d", &n);

    Printf (" Enter Starting vertex ")
    Scanf (" %d ", &s);
    bfs ( graph, vis, s, n );
    return 0;
}
```

Output

Enter no. of vertices : 2
Enter adjacency matrix :
0 1
3 2
Enter staring point : 1
BFS traversal : 0 2 1

Qb. WAP to Traverse a graph using DFS method.

```
void dfs (int graph[MAX][MAX], int vis[MAX], int s, int n)
{
    vis[s] = 1
    printf("%d", s);

    for (int i=0 ; i<n ; i++)
    {
        if (graph[start][i] == 1 && !vis[i])
        {
            dfs(graph, vis, i, n);
        }
    }
}

int main() {

  int graph[MAX][MAX], vis[MAX] = {0}, n, s ;

  printf (" Enter noo. of vertices");
   Scanf ("%d", &n);

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

```
    printf(" Enter the Starting vertex ");
    scanf(" %d ", &s);

    printf(" DFS Traversal ");
    dfs(graph, vis, s, n);

    return 0;
}
```

Output /

Enter no of vertices : 4
Enter adjacenty matrix :

```
0 1 1 0
1 0 1 1
0 1 1 0
1 0 1 0
```

Enter the Starting point : 3
DFS Traversal : 3 0 1 2

CODE :

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

void bfs(int graph[MAX][MAX], int visited[], int n, int start) {
    int queue[MAX], front = -1, rear = -1;
    printf("BFS Traversal starting from vertex %d: ", start);

    queue[++rear] = start;
    visited[start] = 1;

    while (front != rear) {
        int current = queue[++front];
        printf("%d ", current);

        for (int i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

void dfs(int graph[MAX][MAX], int visited[], int n, int start) {
    visited[start] = 1;
    printf("%d ", start);
    for (int i = 0; i < n; i++) {
        if (graph[start][i] == 1 && !visited[i]) {
            dfs(graph, visited, n, i);
        }
    }
}

int isConnected(int graph[MAX][MAX], int n) {
    int visited[MAX] = {0};

    dfs(graph, visited, n, 0);

    for (int i = 0; i < n; i++) {
```

```
        if (!visited[i]) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int graph[MAX][MAX], visited[MAX], n, start;

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

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

    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &start);
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    bfs(graph, visited, n, start);
    printf("--------END-------\n");

    printf("Enter the starting vertex for DFS: ");
    scanf("%d", &start);
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    printf("DFS Traversal starting from vertex %d: ", start);
    dfs(graph, visited, n, start);
    printf("\n");

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

```
    }

    return 0;
}
```

OUTPUT

Output Clear

```
Enter the number of vertices: 4
Enter the adjacency matrix of the graph:
1 0 1 0
1 0 1 1
1 1 1 0
0 1 0 1
Enter the starting vertex for BFS: 3
BFS Traversal starting from vertex 3: 3 1 0 2
--------END-------
Enter the starting vertex for DFS: 2
DFS Traversal starting from vertex 2: 2 0 1 3
    --------END-------

0 2 1 3 The graph is connected.
```