VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SHASHANK SHANTHARAM NAYAK
(1BM23CS313)

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 **SHASHANK SHANTHARAM NAYAK(1BM23CS313),** who is bonafide student of **B.M.S.College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academicrequirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

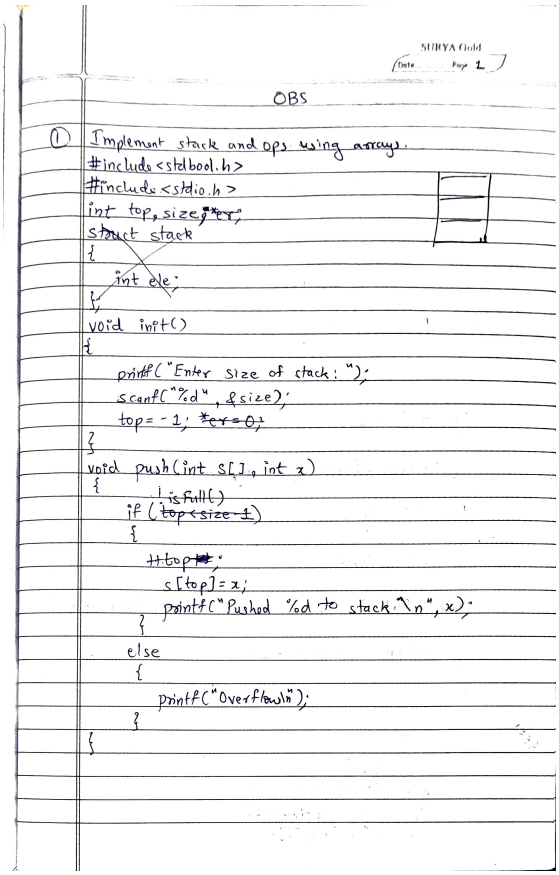
Dr. Namratha M. Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

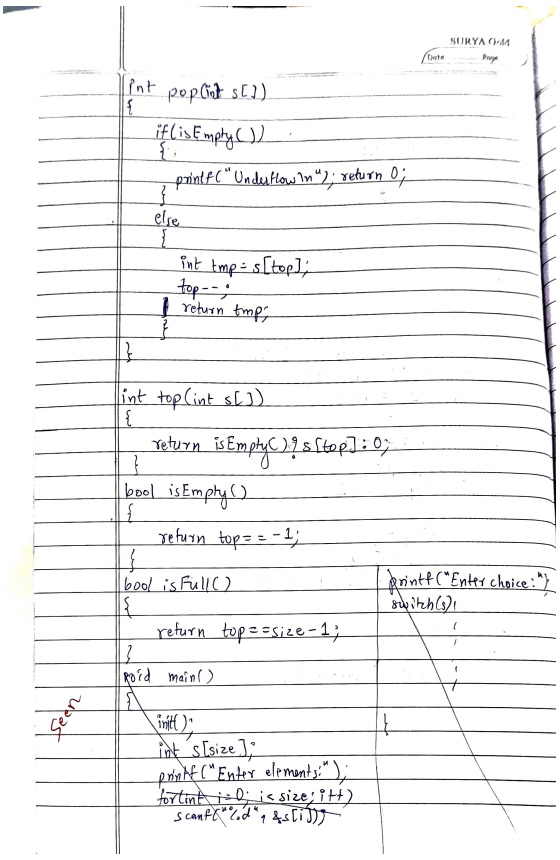
Github Link:
https://github.com/DevelSSN/1BM23CS313-SHASHANK-SHANTHARAM-NAYAK

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



```
OBS
① Implement stack and ops using arrays.
#include <stdbool.h>
#include <stdio.h>
int top, size, *arr;
struct stack
{
    int ele;
};
void init()
{
    printf("Enter size of stack: ");
    scanf("%d", &size);
    top = -1; *arr = 0;
}
void push(int s[], int x)
{
    isfull()
    if (top < size-1)
    {
        ++top;
        s[top] = x;
        printf("Pushed %d to stack.\n", x);
    }
    else
    {
        printf("Overflow\n");
    }
}
```



```
int pop(int s[])
{
    if(isEmpty())
    {
        printf("Underflow\n"); return 0;
    }
    else
    {
        int tmp = s[top];
        top--;
        return tmp;
    }
}

int top(int s[])
{
    return isEmpty()? s[top] : 0;
}
bool isEmpty()
{
    return top == -1;
}
bool isFull()
{
    return top == size-1;
}
void main()
{
    init();
    int s[size];
    printf("Enter elements:");
    for(int i=0; i<size; i++)
        scanf("%d", &s[i]);
}
printf("Enter choice:");
switch(c)
```

SURYA Gold Date Page 2

```
void display(int s[])
{
    if (!isEmpty())
    {
        printf("Elements are: \t");
        for(int i=0; i<=top; i++)
        {
            printf("%d \t", s[i]);
        } printf("\n");
    }
    else
    {
        printf("No element to display\n");
    }
}
void main()
{
    init();
    int s[size];
    char c, choice;
    do
    {
        printf("\n\n Operations: \n");
        printf("push(p) \t\t pop(P) \n Top(t) \t\t Display(d) \n
            Empty(e) \t Full(f) \n 0 to exit \n Enter choice:");
        // Flush
        while((c = getchar()) != '\n' && c != EOF) { }
        scanf("%c", &choice);

        switch(choice)
        {
```

SURYA Gold Date Page

```
            case 'p': int x;
                printf("Enter element: ");
                scanf("%d", &x);
                push(s, x);
                break;
            case 'P': printf("Element %d popped\n", pop(s));
                break;
            case 'e': if(isEmpty())
                    printf("Empty stack\n");
                else
                    printf("Not Empty\n");
                break;
            case 't': printf("Top element: %d\n", Top(s));
                break;
            case 'f': if(isFull())
                    printf("Stack is full\n");
                else
                    printf("Stack is not full\n");
                break;
            case 'd': display(s);
                break;
            case '0': break;
            default: break;
        }
    }while(choice != '0');
    printf("\n Exiting...\n");
}
```

SURYA Gold
Date Page 3

Output:

Enter the size of stack: 3

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: p
Enter element: 4
Pushed 4 to stack

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: p
Enter element: 5
Pushed 5 to stack

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: p
Enter element: 6
Pushed 6 to stack

SURYA Gold
Date Page

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: p
Enter element: 7
Overflow

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: P
Element 6 popped

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: P
Element 5 popped

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: P
Element 4 popped

SURYA Gold
Date Page 4

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: P
Underflow
Element 0 popped

Operations:
push(p) pop(P)
Top(t) Display(d)
Empty(e) Full(F)
0 to exit
Enter choice: 0

Exiting...

olp Seen

Code:

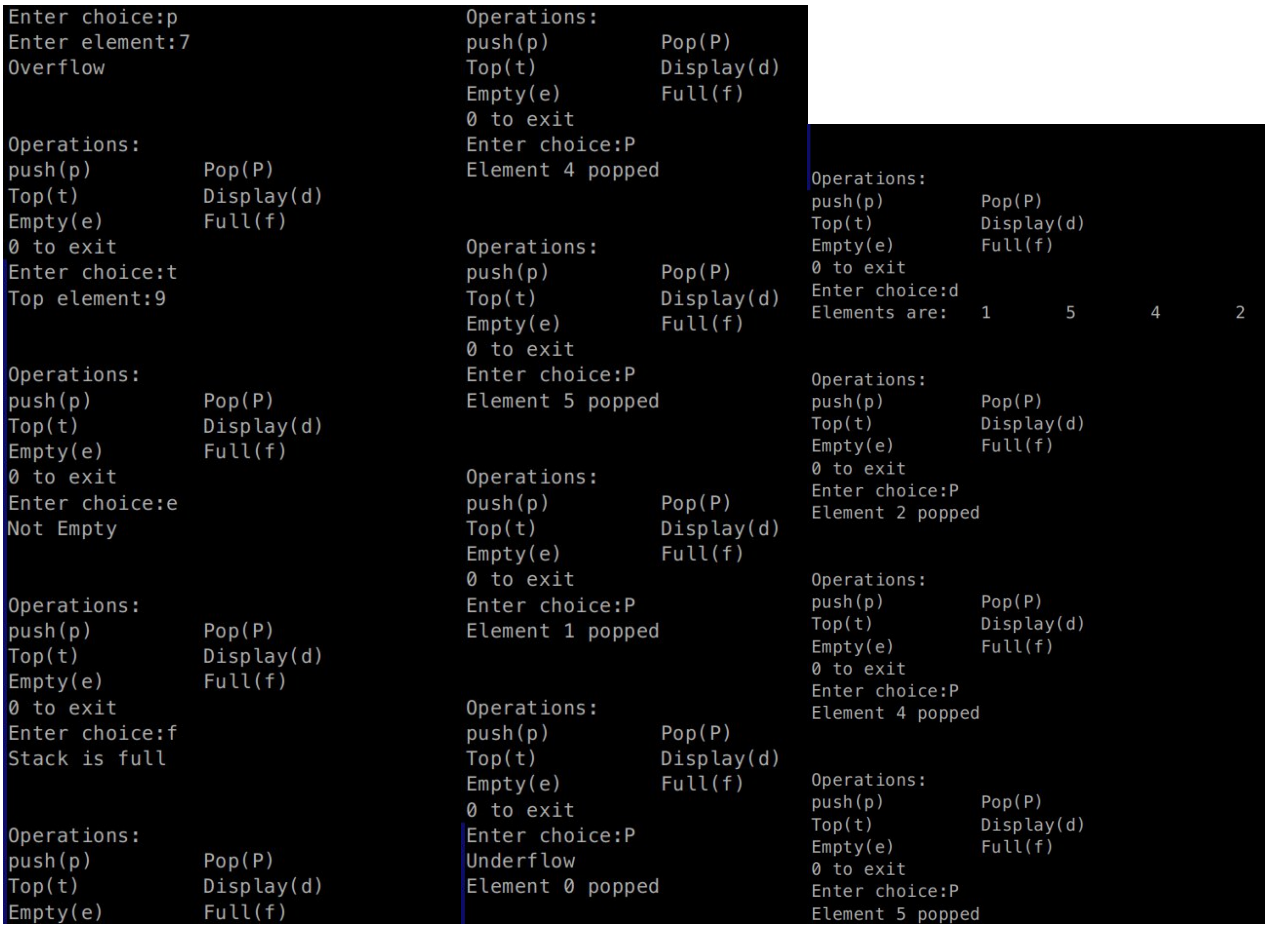
```
#include <stdbool.h>
#include <stdio.h>
int top, size;

void init()
{
	printf("Enter the size of stack:");
	scanf("%d",&size);
	top=-1;
}

bool isEmpty()
{return top==-1;}

bool isFull()
{return top==size-1;}

void push(int s[], int x)
{
	if(!isFull())
	{
		++top;
		s[top]=x;
		printf("Pushed %d to stack\n",x);
		printf("%d",top);
	}
	else
	{
		printf("Overflow\n");
	}
}

int pop(int s[])
{
	if(isEmpty())
	{
		printf("Underflow\n");
		return 0;
	}
	else
	{
		int tmp=s[top];
		top--;
		return tmp;
	}
}
```

```
int Top(int s[])
{return s[top];}

void display(int s[])
{
	if(!isEmpty())
	{
		printf("Elements are:\t");
		for(int i=0;i<=top;i++)
			printf("%d\t", s[i]);
		printf("\n");
	}
	else
		printf("No element to display\n");
}

void main()
{
	init();
	int s[size];
	char choice, c;
	do{
	printf("\n\nOperations:\n");
	printf("push(p)\t\tPop(P)\nTop(t)\t\tDisplay(d)\nEmpty(e)\tFull(f)\n0 to exit\nEnter choice:");
	// Flush stdin
	while ((c = getchar()) != '\n' && c != EOF) { }
	// TODO input here
	scanf("%c",&choice);
	switch(choice)
	{
		case 'p':int x;
			printf("Enter element:");
			scanf("%d",&x);
			push(s,x);
			break;
		case 'P':printf("Element %d popped\n",pop(s));
			break;
		case 'e':if(isEmpty())
				printf("Empty Stack\n");
			else
				printf("Not Empty\n");
			break;
		case 't':printf("Top element:%d\n",Top(s));
			break;
		case 'f':if(isFull())
				printf("Stack is full\n");
			else
```

```
                        printf("Stack is not full\n");
                    break;
                case 'd':display(s);
                    break;
                case '0': break;
                default: break;
        }
        }while(choice!='0');
        printf("\nExiting...\n");
}
```

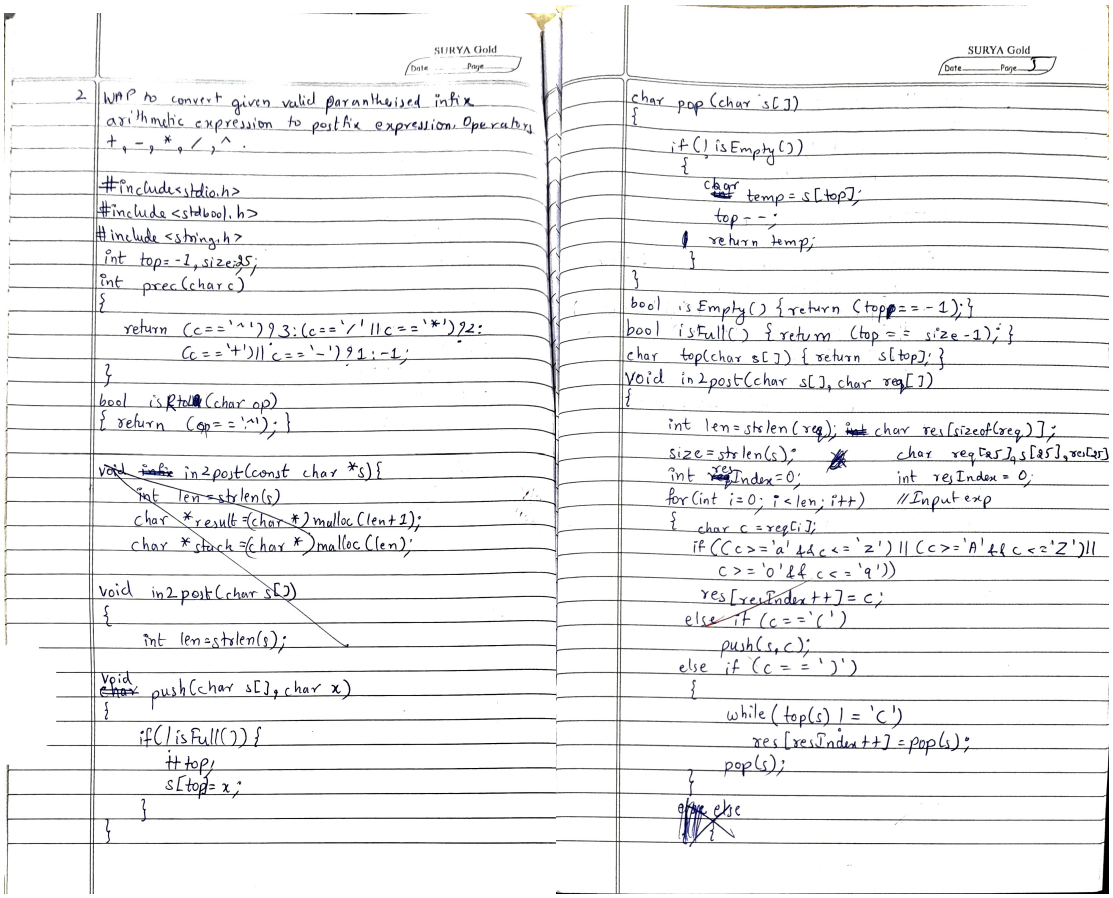
OUTPUT

```
❯ ./stack
Enter the size of stack:5

Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
Enter choice:p
Enter element:1
Pushed 1 to stack
0

Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
Enter choice:p
Enter element:5
Pushed 5 to stack
1

Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
Enter choice:p
Enter element:4
Pushed 4 to stack
2

Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
Enter choice:p
Enter element:2
Pushed 2 to stack
3

Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
Enter choice:p
Enter element:9
Pushed 9 to stack
4

Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
Enter choice:p
Enter element:7
Overflow


Operations:
push(p)         Pop(P)
Top(t)          Display(d)
Empty(e)        Full(f)
0 to exit
```


Enter choice:p
Enter element:7
Overflow
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:t
Top element:9
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:e
Not Empty
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:f
Stack is full
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Element 4 popped
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Element 5 popped
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Element 1 popped
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Underflow
Element 0 popped
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:d
Elements are: 1 5 4 2
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Element 2 popped
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Element 4 popped
Operations:
push(p) Pop(P)
Top(t) Display(d)
Empty(e) Full(f)
0 to exit
Enter choice:P
Element 5 popped

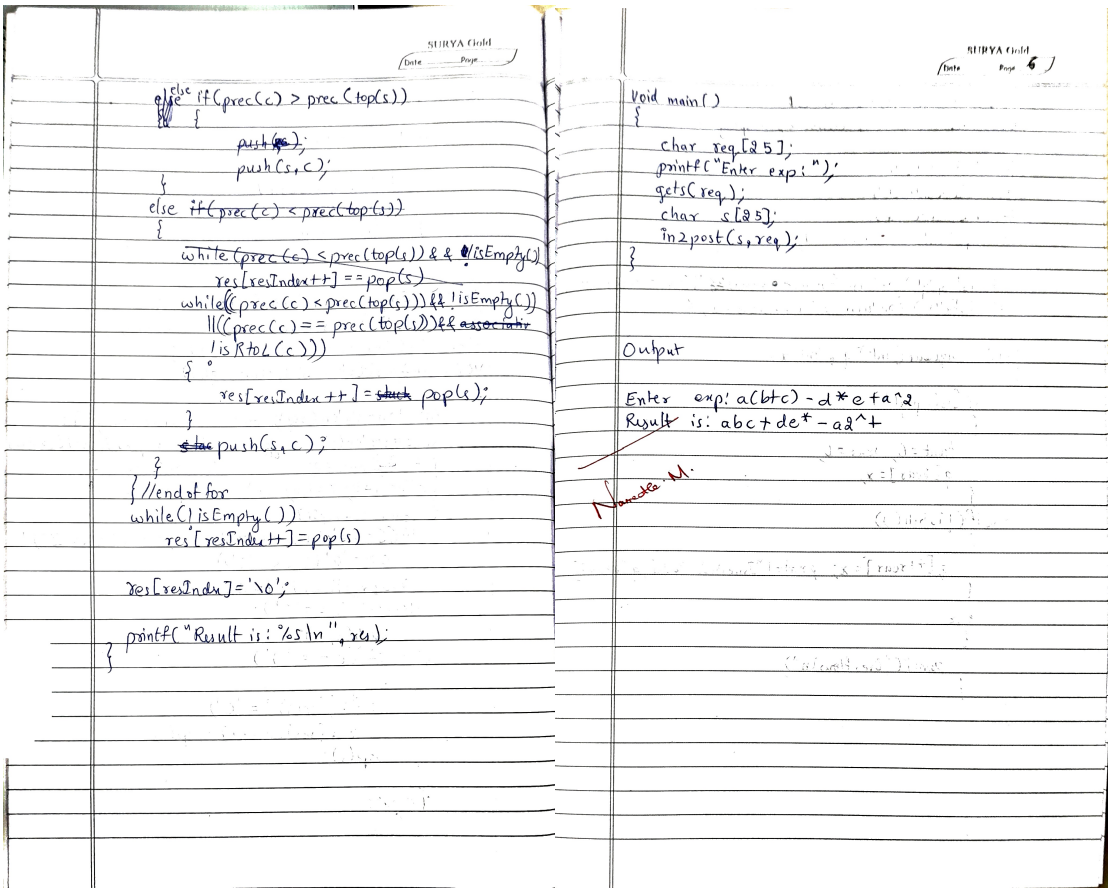
Program 2

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).



```
2 WAP to convert given valid paranthesised infix
arithmetic expression to postfix expression. Operators
+, -, *, /, ^.

#include<stdio.h>
#include<stdbool.h>
#include<string.h>
int top=-1, size=25;
int prec(char c)
{
    return (c=='^')?3:(c=='/'||c=='*')?2:
           (c=='+'||c=='-')?1:-1;
}
bool isRtol(char op)
{ return (op=='^'); }

void infix in2post(const char *s){
    int len=strlen(s)
    char *result=(char *)malloc(len+1);
    char *stack=(char *)malloc(len);

void in2post(char s[])
{
    int len=strlen(s);

Void
Char push(char s[], char x)
{
    if(!isFull()){
        ++top;
        s[top]=x;
    }
}
```

```
SURYA Gold
Date Page 3

char pop(char s[])
{
    if(!isEmpty())
    {
        char temp=s[top];
        top--;
        return temp;
    }
}
bool isEmpty(){return (top==-1);}
bool isFull(){return (top==size-1);}
char top(char s[]){return s[top];}
void in2post(char s[], char req[])
{
    int len=strlen(req); char res[sizeof(req)];
    size=strlen(s);                char req[85],s[85],res[85]
    int resIndex=0;                int resIndex=0;
    for(int i=0; i<len; i++)       //Input exp
    {   char c=req[i];
        if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||
           c>='0'&&c<='9'))
            res[resIndex++]=c;
        else if(c=='(')
            push(s,c);
        else if(c==')')
        {
            while(top(s)!='(')
                res[resIndex++]=pop(s);
            pop(s);
        }
        else
```



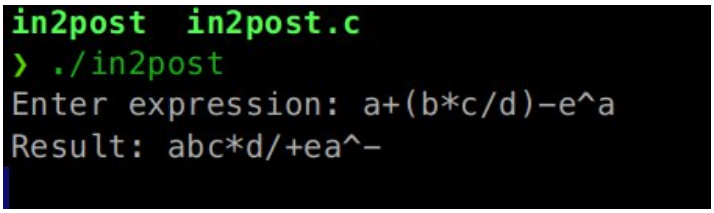
Code:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
int top=-1, size=25;

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

bool isEmpty(){return (top==-1);}
bool isFull(){return (top==size-1);}

bool isRtoL(char op){return (op=='^');}

char Top(char s[]){if (!isEmpty()) {
      return s[top];
   }
   return '\0';  }
void push(char s[], char x)
{
        if (!isFull()) {
```

```
        ++top;
        s[top] = x;
    }
}

char pop(char s[]){
          if (!isEmpty()) {
        return s[top--];
    }
    return '\0';
}

void in2post(){
    char req[25];
    char s[25];  // Stack for operators
    char res[25];  // Result array
    int resIndex = 0;

    printf("Enter expression: ");
    fgets(req, sizeof(req), stdin);

    int len = strlen(req);
    memset(res, 0, sizeof(res));  // Clear result array

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

        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')) {
            res[resIndex++] = c;  // Append operand to result
        } else if (c == '(') {
            push(s, c);  // Push '(' to stack
        } else if (c == ')') {
            while (!isEmpty() && Top(s) != '(') {
                res[resIndex++] = pop(s);
            }
            pop(s);  // Pop '(' from stack
        } else if (prec(c) > prec(Top(s))) {
            push(s, c);  // Push operator to stack
        } else {
            while (!isEmpty() && ((prec(c) < prec(Top(s))) || (prec(c) == prec(Top(s)) && !isRtoL(c))))
{
                res[resIndex++] = pop(s);
            }
            push(s, c);
        }
    }

    // Clear stack
```

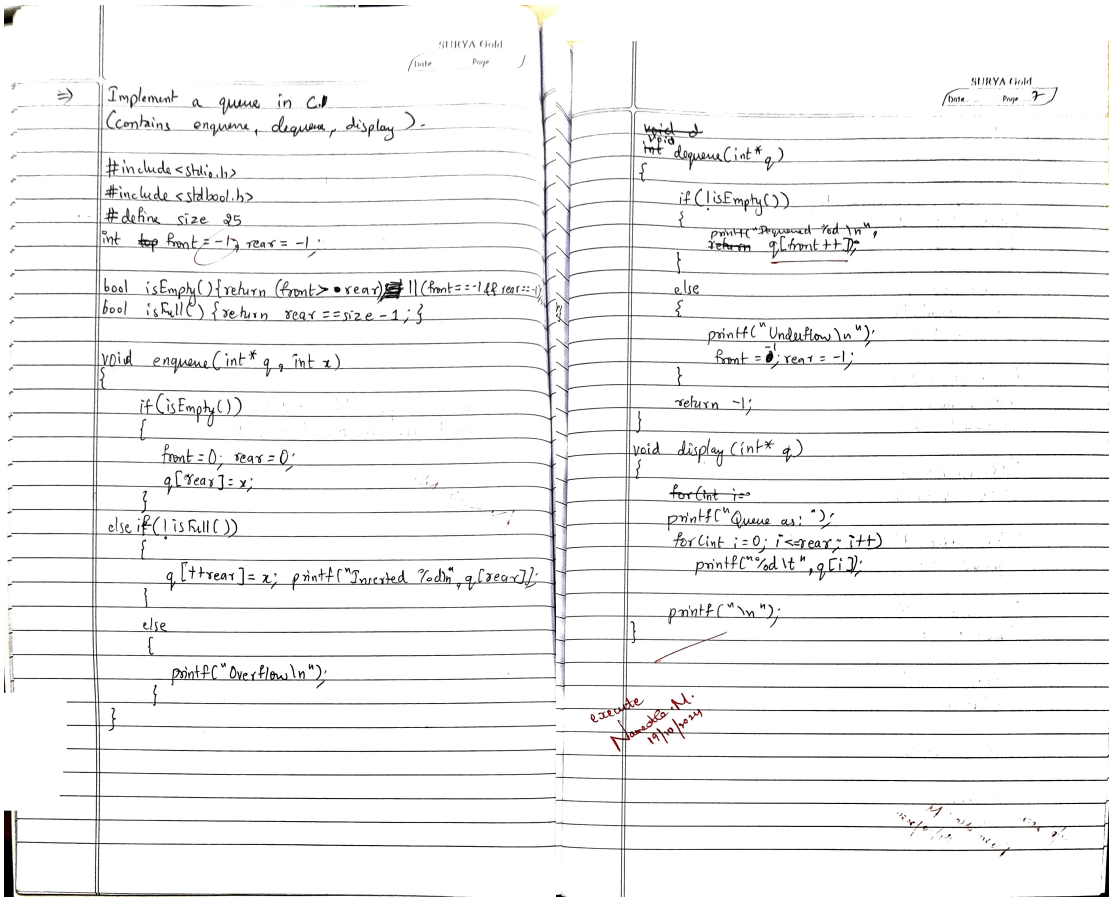
```
    while (!isEmpty()) {
        res[resIndex++] = pop(s);
    }

    res[resIndex] = '\0';  // Null-terminate the result string
    printf("Result: %s\n", res);
}

void main()
{
        in2post();
}
```

OUTPUT

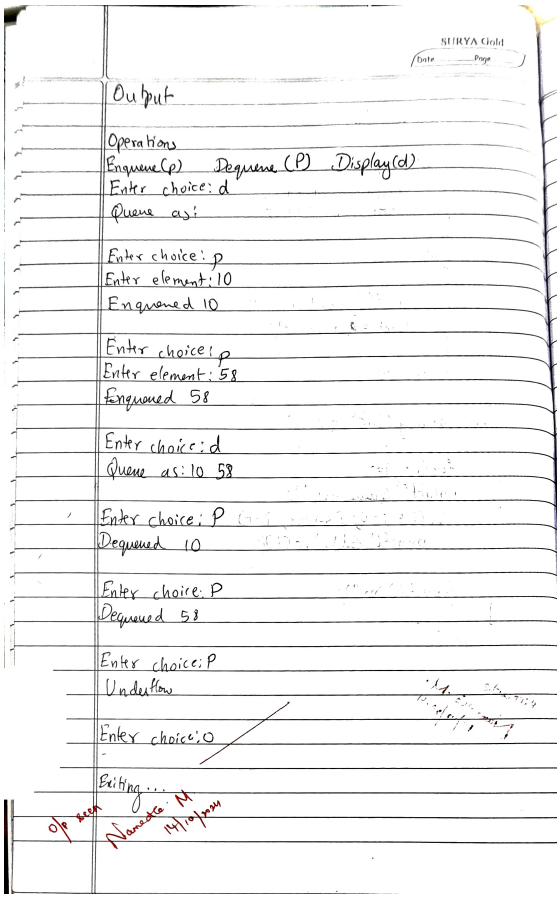


Program 3:

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.



```
Implement a queue in C
(Contains enqueue, dequeue, display).

#include<stdio.h>
#include<stdbool.h>
#define size 25
int front=-1, rear=-1;

bool isEmpty(){return (front>rear)||(front==-1&&rear==-1);}
bool isFull(){return rear==size-1;}

void enqueue(int* q, int x)
{
    if(isEmpty())
    {
        front=0; rear=0;
        q[rear]=x;
    }
    else if(!isFull())
    {
        q[++rear]=x; printf("Inserted %d\n",q[rear]);
    }
    else
    {
        printf("Overflow\n");
    }
}

int dequeue(int* q)
{
    if(!isEmpty())
    {
        printf("Deleted %d\n",q[front++]);
    }
    else
    {
        printf("Underflow\n");
        front=0; rear=-1;
    }
    return -1;
}
void display(int* q)
{
    printf("Queue as: ");
    for(int i=0; i<=rear; i++)
        printf("%d\t",q[i]);

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



Code:

```
#include <stdio.h>
#include <stdbool.h>
#define size 5
int front=-1, rear=-1;
bool isEmpty(){return (front>rear)||(front==-1&&rear==-1);}
bool isFull(){return rear==size-1;}

void enqueue(int* q, int x)
{
	if(isEmpty())
	{
		front=0; rear=0;
		q[rear]=x;
	}
	else if(!isFull())
	{
		q[++rear]=x;
		printf("Enqueued %d\n",q[rear]);
	}
	else
```

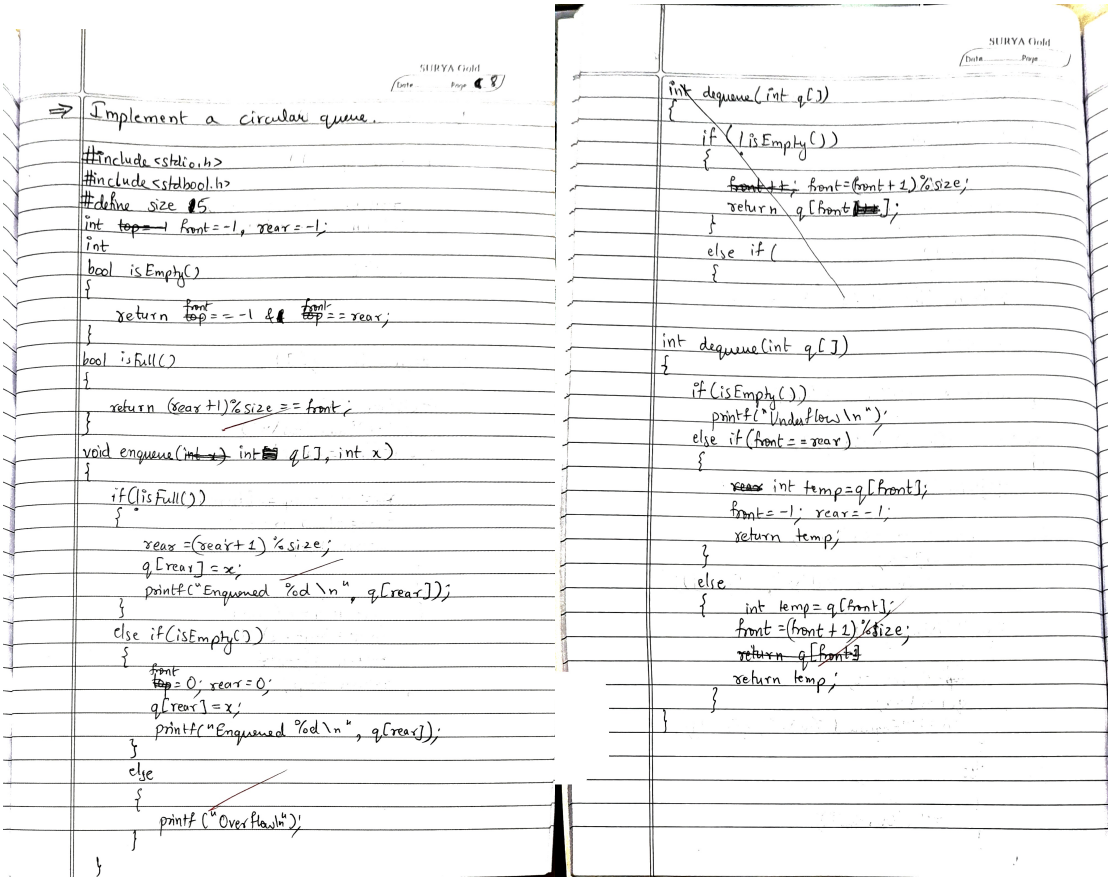
```
	{
		printf("Overflow\n");
	}
}

void dequeue(int* q)
{
	if(!isEmpty())
	{
		printf("Dequeued %d\n", q[front++]);
	}
	else
	{
		printf("Underflow\n");
		front=-1;rear=-1;
	}
}
void display(int* q)
{
	printf("Queue as:");
	for(int i=0;i<=rear;i++)
		printf("%d\t",q[i]);
	printf("\n");
}

void main()
{
	int q[size];
	char choice, c;
	printf("\n\nOperations:\n");
	printf("Enqueue(p)\t\tDequeue(P)\t\tDisplay(d)\n0 to exit\n");
	do{
		// TODO input here
		printf("\nEnter choice:");
		scanf("%c",&choice);
	switch(choice)
	{
		case 'p':int x;
			printf("Enter element:");
			scanf("%d",&x);
			enqueue(q,x);
			break;
		case 'P':dequeue(q);
			break;
		case 'e':if(isEmpty())
				printf("Empty Stack\n");
			else
```

```
                printf("Not Empty\n");
            break;
        case 'f':if(isFull())
                printf("Stack is full\n");
            else
                printf("Stack is not full\n");
            break;
        case 'd':display(q);
            break;
        case '0': break;
        default: break;
    }
    // Flush stdin
    if((c=getchar())=='\n'||c==EOF){}
    }while(choice!='0');
    printf("\nExiting...\n");
}
```

OUTPUT

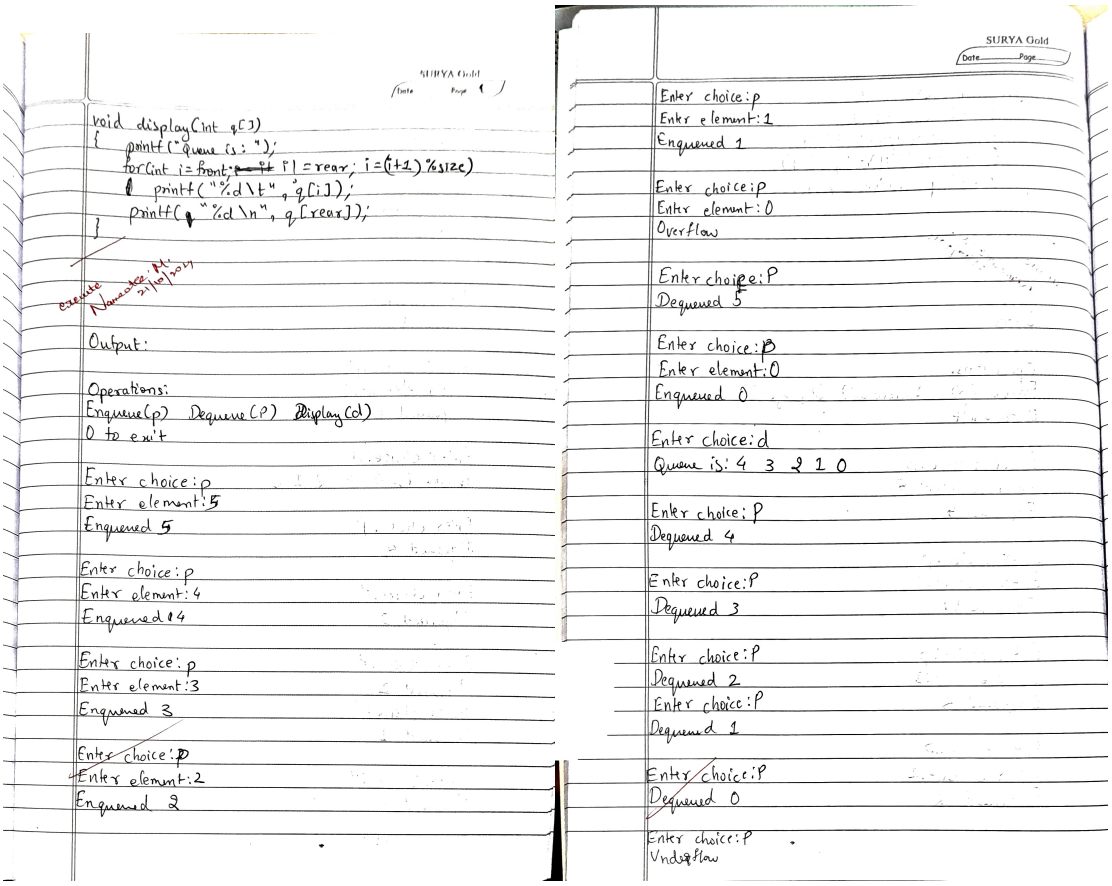
```
❯ ./queue

Operations:
Enqueue(p)          Dequeue(P)          Display(d)
0 to exit

Enter choice:p
Enter element:10

Enter choice:p
Enter element:9
Enqueued 9

Enter choice:p
Enter element:8
Enqueued 8

Enter choice:p
Enter element:7
Enqueued 7

Enter choice:p
Enter element:6
Enqueued 6

Enter choice:p
Enter element:5
Overflow

Enter choice:P
Dequeued 10

Enter choice:p
Enter element:6
Overflow

Enter choice:P
Dequeued 10

Enter choice:p
Enter element:6
Overflow

Enter choice:d
Queue as:10     9       8       7       6

Enter choice:P
Dequeued 9

Enter choice:P
Dequeued 8

Enter choice:P
Dequeued 7

Enter choice:P
Dequeued 6

Enter choice:P
Underflow

Enter choice:P
Underflow

Enter choice:0

Exiting...
```

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.



```
⇒ Implement a circular queue.

#include <stdio.h>
#include <stdbool.h>
#define size 5
int front=-1, rear=-1;
int
bool isEmpty()
{
    return front == -1 && front == rear;
}
bool isFull()
{
    return (rear+1)%size == front;
}
void enqueue(int q[], int x)
{
    if(!isFull())
    {
        rear =(rear+1) %size;
        q[rear] = x;
        printf("Enqueued %d\n", q[rear]);
    }
    else if(isEmpty())
    {
        front = 0; rear = 0;
        q[rear] = x;
        printf("Enqueued %d\n", q[rear]);
    }
    else
    {
        printf("Overflow\n");
    }
}
```

```
int dequeue(int q[])
{
    if (!isEmpty())
    {
        front = (front+1)%size;
        return q[front];
    }
    else if(
    {
    }

int dequeue(int q[])
{
    if(isEmpty())
        printf("Underflow\n");
    else if(front == rear)
    {
        int temp=q[front];
        front = -1; rear = -1;
        return temp;
    }
    else
    {   int temp = q[front];
        front =(front+1)%size;
        return temp;
    }
}
```



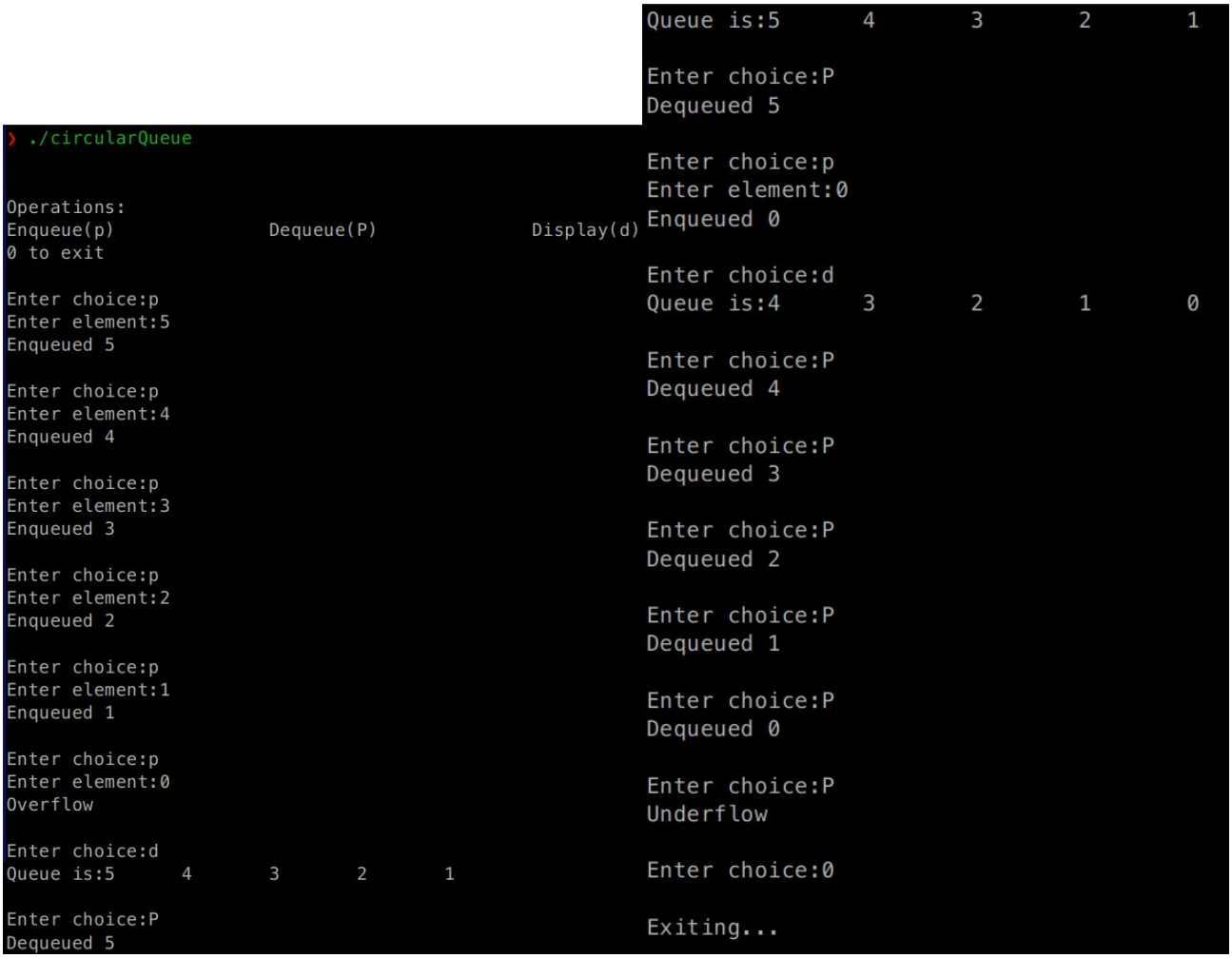
Code:

```
#include <stdio.h>
#include <stdbool.h>
#define size 5
int front=-1, rear=-1;
bool isEmpty(){return front==-1&rear==front;}
bool isFull(){return (rear+1)%size==front;}

void enqueue(int q[], int x)
{
	if(isEmpty())
	{
		front=0; rear=0;
		q[rear]=x;
		printf("Enqueued %d\n",q[rear]);
	}
	else if(!isFull())
	{
		rear=(rear+1)%size;
		q[rear]=x;
```

```
		printf("Enqueued %d\n",q[rear]);
	}
	else
	{
		printf("Overflow\n");
	}
}

int dequeue(int q[])
{
	if(isEmpty())
	{
		printf("Underflow\n");
	}
	else if(front==rear)
	{
		int temp=q[front];
		front=-1;rear=-1;
		return temp;
	}
	else
	{
		int temp=q[front];
		front=(front+1)%size;
		return temp;
	}
}
void display(int* q)
{
	printf("Queue is:");
	for(int i=front;i!=rear;i=(i+1)%size)
		printf("%d\t",q[i]);
	printf("%d\n",q[rear]);
}

void main()
{
	int q[size];
	char choice, c;
	printf("\n\nOperations:\n");
	printf("Enqueue(p)\t\tDequeue(P)\t\tDisplay(d)\n0 to exit\n");
	do{
		// TODO input here
		printf("\nEnter choice:");
		scanf("%c",&choice);
```

```
        switch(choice)
        {
                case 'p':int x;
                        printf("Enter element:");
                        scanf("%d",&x);
                        enqueue(q,x);
                        break;
                case 'P':if(isEmpty())
                                        printf("Underflow\n");
                                else
                                        printf("Dequeued %d\n",dequeue(q));
                                break;
                case 'd':display(q);
                        break;
                case '0': break;
                default: break;
        }
        // Flush stdin
        if((c=getchar())=='\n'||c==EOF){}
        }while(choice!='0');
        printf("\nExiting...\n");

}
```

OUTPUT


❯ ./circularQueue
Operations:
Enqueue(p) Dequeue(P) Display(d)
0 to exit
Enter choice:p
Enter element:5
Enqueued 5
Enter choice:p
Enter element:4
Enqueued 4
Enter choice:p
Enter element:3
Enqueued 3
Enter choice:p
Enter element:2
Enqueued 2
Enter choice:p
Enter element:1
Enqueued 1
Enter choice:p
Enter element:0
Overflow
Enter choice:d
Queue is:5 4 3 2 1
Enter choice:P
Dequeued 5
Queue is:5 4 3 2 1
Enter choice:P
Dequeued 5
Enter choice:p
Enter element:0
Enqueued 0
Enter choice:d
Queue is:4 3 2 1 0
Enter choice:P
Dequeued 4
Enter choice:P
Dequeued 3
Enter choice:P
Dequeued 2
Enter choice:P
Dequeued 1
Enter choice:P
Dequeued 0
Enter choice:P
Underflow
Enter choice:0
Exiting...

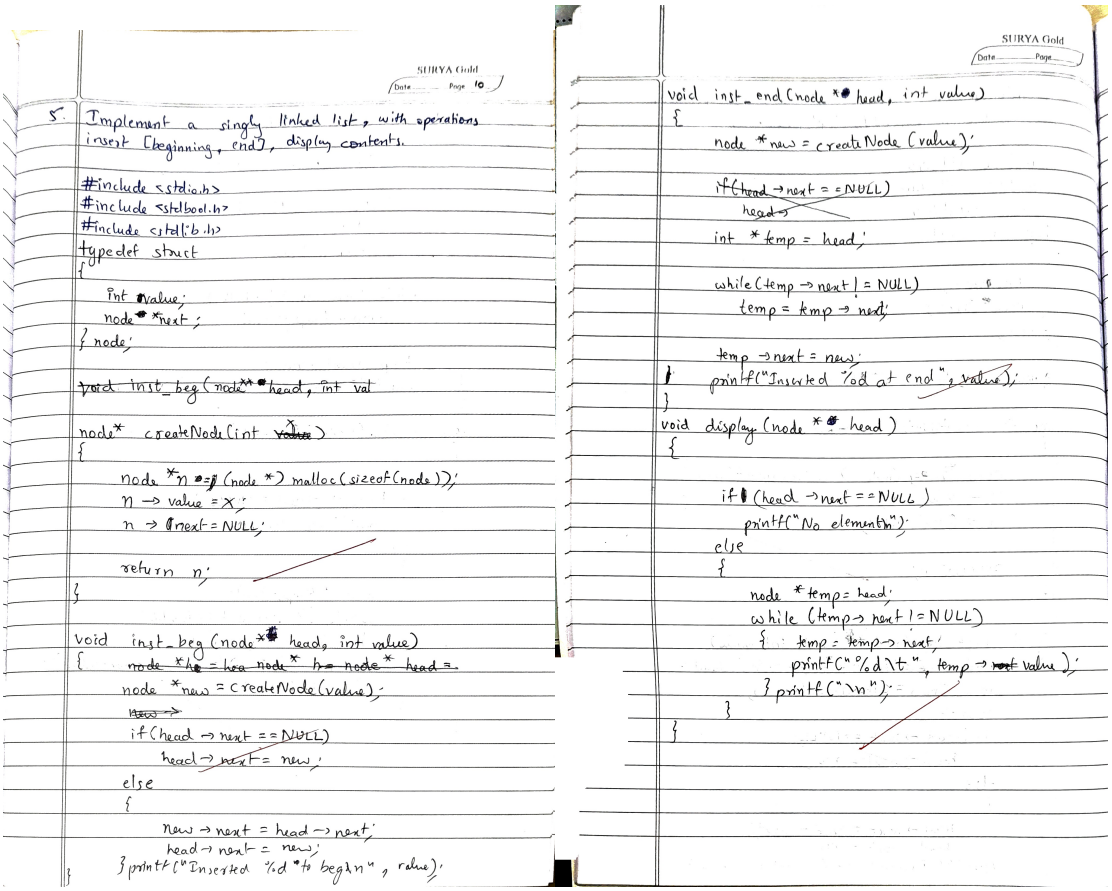
Problem 4:

WAP to Implement Singly Linked List with following operations:

a) Create a linked list.

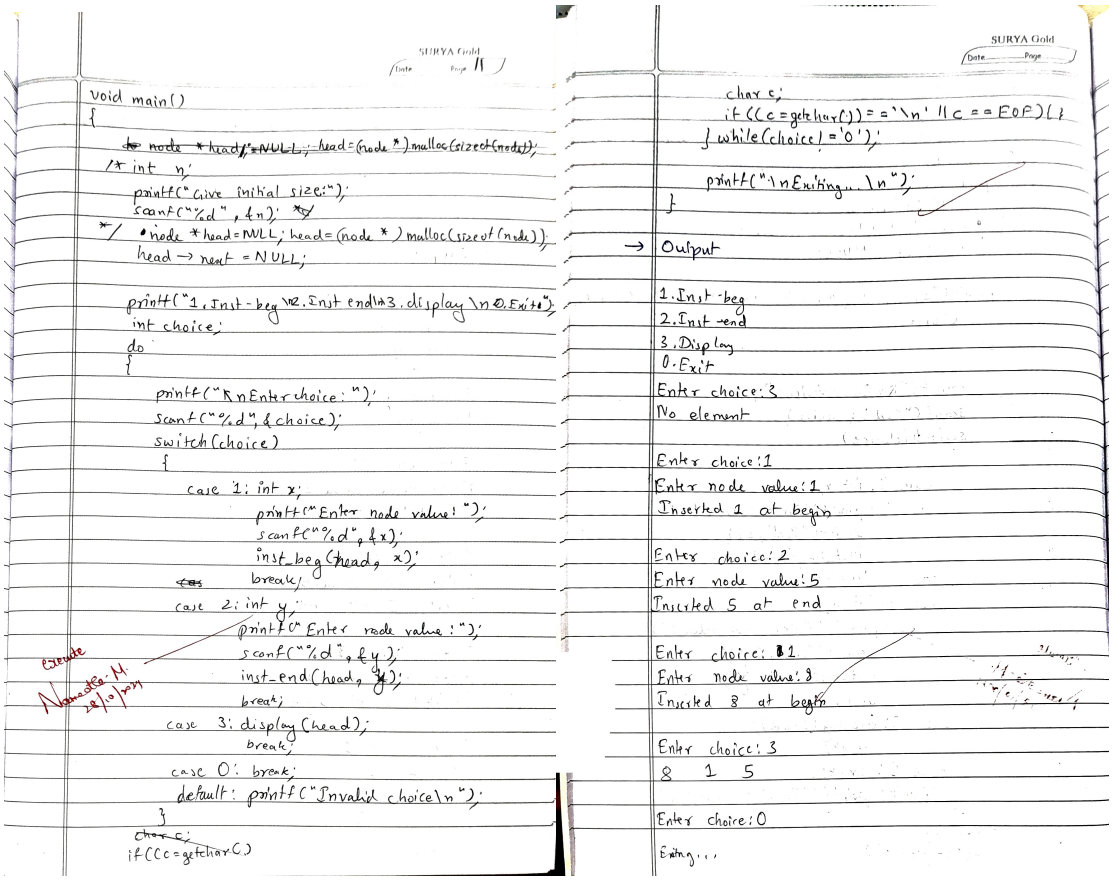
b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.



```
5. Implement a singly linked list, with operations
insert (beginning, end), display contents.

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
typedef struct
{
    int value;
    node *next;
} node;

node* createNode(int x)
{
    node *n = (node *) malloc(sizeof(node));
    n -> value = x;
    n -> next = NULL;

    return n;
}

void inst_beg(node *head, int value)
{
    node *new = createNode(value);
    if(head -> next == NULL)
        head -> next = new;
    else
    {
        new -> next = head -> next;
        head -> next = new;
    } printf("Inserted %d at begin\n", value);
```

```
void inst_end(node *head, int value)
{
    node *new = createNode(value);

    if(head -> next == NULL)
        head->
    int *temp = head;

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

    temp -> next = new;
    printf("Inserted %d at end", value);
}
void display(node *head)
{

    if(head -> next == NULL)
        printf("No elements");
    else
    {
        node *temp = head;
        while(temp -> next != NULL)
        { temp = temp -> next;
            printf("%d\t", temp -> value);
        } printf("\n");
    }
}
```



Code:

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
typedef struct node node;
struct node
{
	int value;
	struct node *next;
};

node* createNode(int x)
{
	node *n = (node*)malloc(sizeof(node));
	n->value=x;
	n->next=NULL;

	return n;
}
```

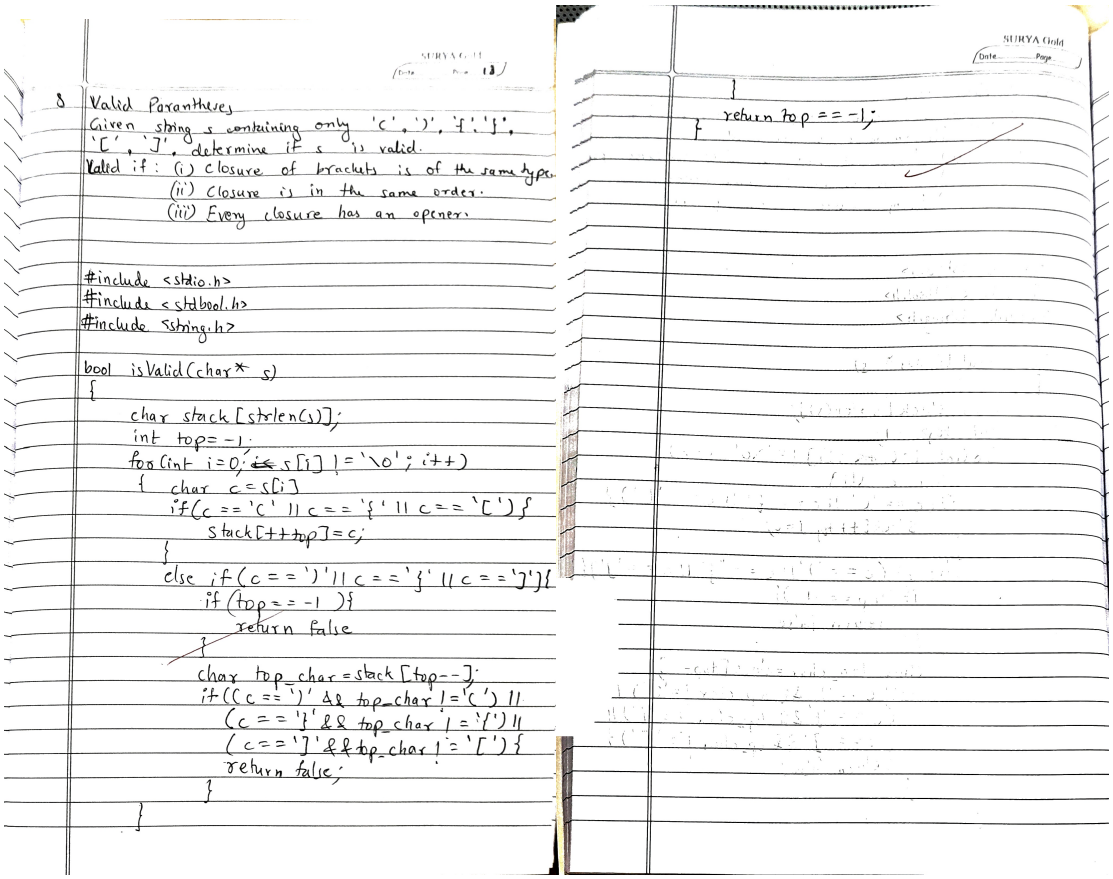
```
void inst_beg(node* head, int value)
{
	node *new = createNode(value);

	if(head->next == NULL)
		head->next = new;
	else
	{
		new->next = head->next;
		head->next = new;
	}
	printf("Inserted %d at begin\n",value);
}

void inst_end(node* head, int value)
{
	node *new = createNode(value);

	node *temp = head;

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

	temp->next = new;
	printf("Inserted %d at end\n",value);
}

void display(node* head)
{
	node *temp= head;
	if(head->next==NULL){
		printf("No element\n");
		return;
	}
	while(temp->next!=NULL)
	{
		temp = temp->next;
		printf("%d\t",temp->value);
	}
	printf("\n");
}

void main()
{
	node *head=NULL;
```

```
        head = (node *) malloc(sizeof(node));
        head->next = NULL;

        printf("1.Inst-beg\n2.Inst-end\n3.Display\n0.Exit");
        int choice;
        char c;
        do
        {
                printf("\nEnter choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1:int x;
                                printf("Enter node value:");
                                scanf("%d",&x);
                                inst_beg(head,x);
                                break;
                        case 2:int y;
                               printf("Enter node value:");
                               scanf("%d",&y);
                               inst_end(head,y);
                                break;
                        case 3:display(head);
                                break;
                        case 0:break;
                        default:printf("Invalid choice\n");
                }
                if((c=getchar())=='\n'||c==EOF){}
        }while(choice!=0);
        printf("\nExiting...\n");
}
```

OUTPUT

```
❯ ./linkedList
1.Inst-beg
2.Inst-end
3.Display
0.Exit
Enter choice:1
Enter node value:1
Inserted 1 at begin

Enter choice:2
Enter node value:5
Inserted 5 at end

Enter choice:1
Enter node value:8
Inserted 8 at begin

Enter choice:3
8       1       5

Enter choice:0

Exiting...
```

Program - Leetcode platform – Valid Parantheses

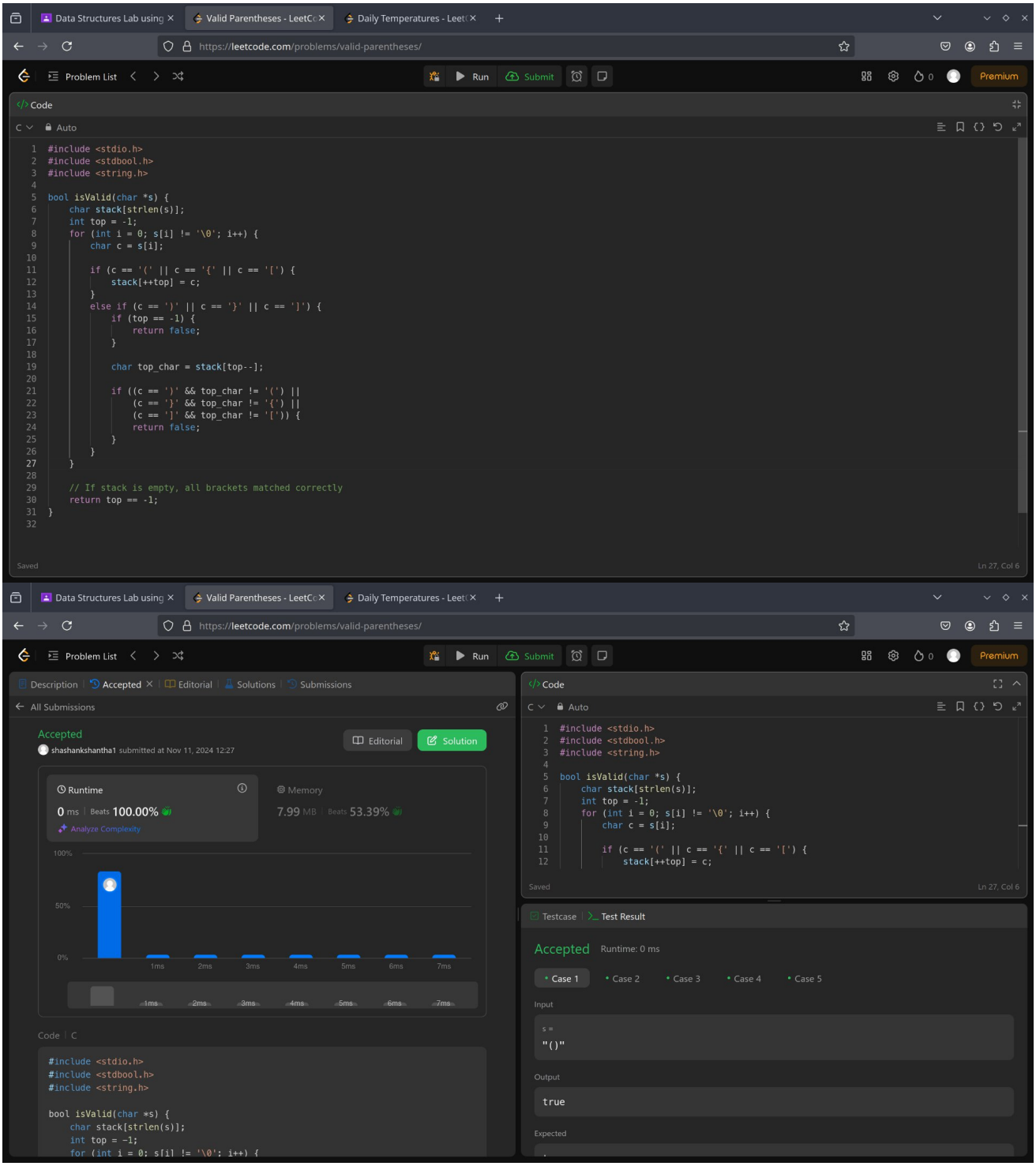


8 Valid Paranthesis
Given string s containing only '(', ')', '{', '}', '[', ']', determine if s is valid.
Valid if: (i) closure of brackets is of the same type.
(ii) closure is in the same order.
(iii) Every closure has an opener.

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

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

```
    return top == -1;
}
```

OUTPUT


Data Structures Lab using
Valid Parentheses - LeetCo
Daily Temperatures - Leet
https://leetcode.com/problems/valid-parentheses/
Problem List
Run
Submit
Premium
Code
Auto
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
bool isValid(char *s) {
char stack[strlen(s)];
int top = -1;
for (int i = 0; s[i] != '\0'; i++) {
char c = s[i];
if (c == '(' || c == '{' || c == '[') {
stack[++top] = c;
}
else if (c == ')' || c == '}' || c == ']') {
if (top == -1) {
return false;
}
char top_char = stack[top--];
if ((c == ')' && top_char != '(') ||
(c == '}' && top_char != '{') ||
(c == ']' && top_char != '[')) {
return false;
}
}
}
// If stack is empty, all brackets matched correctly
return top == -1;
}
Saved
Ln 27, Col 6
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted
shashankshantha1 submitted at Nov 11, 2024 12:27
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
7.99 MB | Beats 53.39%
100%
50%
0%
1ms
2ms
3ms
4ms
5ms
6ms
7ms
Code | C
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Case 4
Case 5
Input
s =
"()"
Output
true
Expected

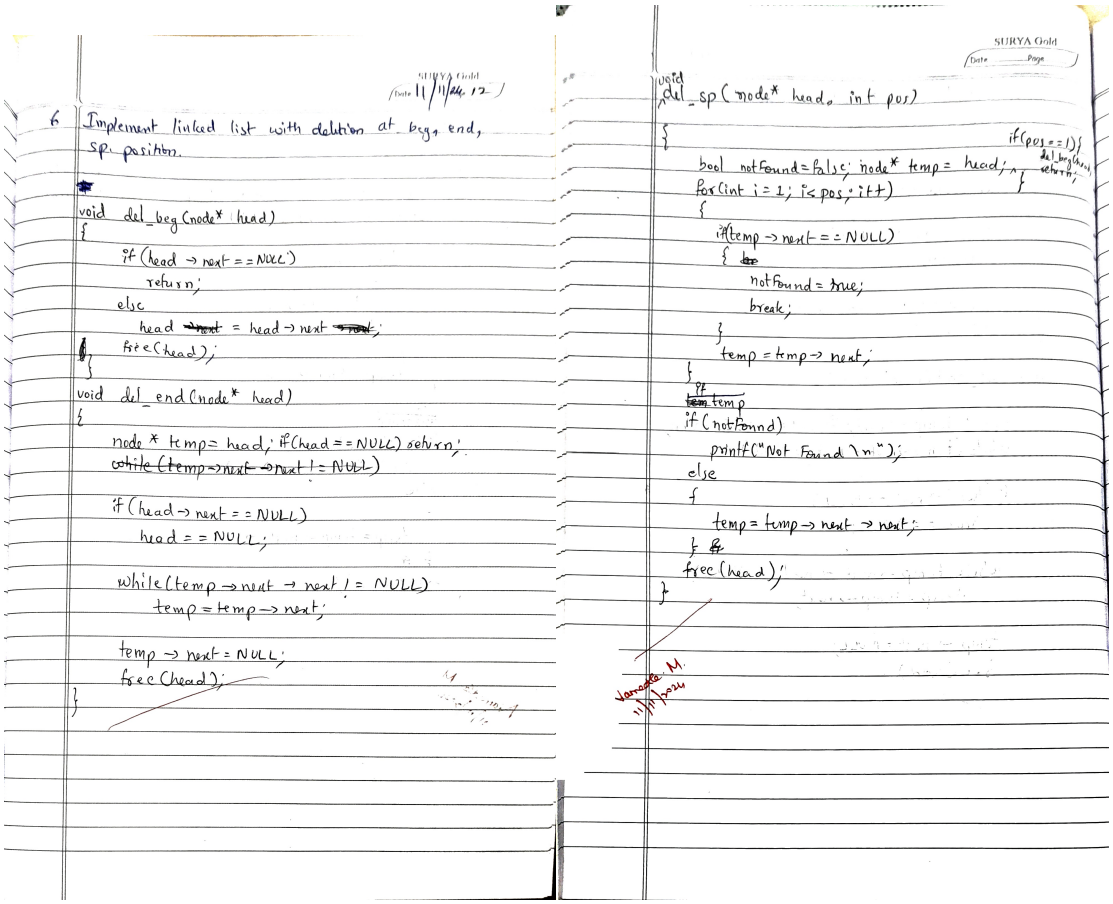
Program 5:

WAP to Implement Singly Linked List with following operations:

a) Create a linked list.

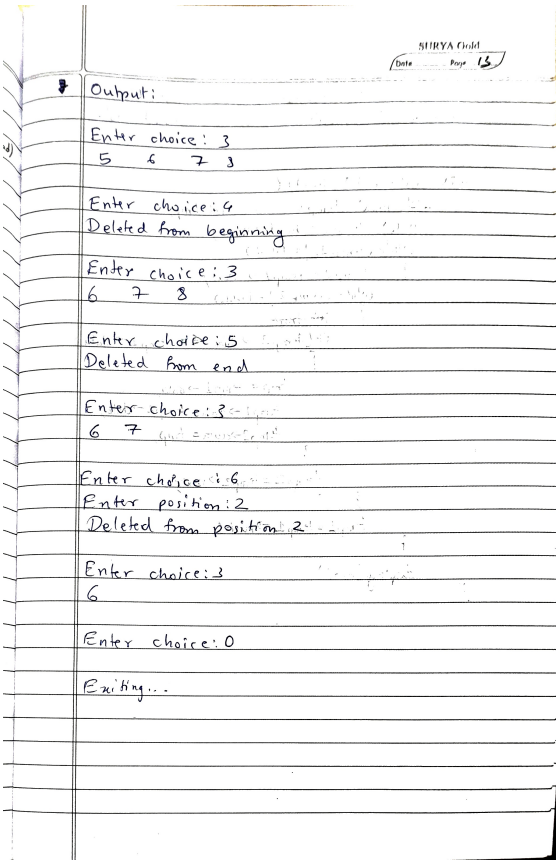
b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.



```
6 Implement linked list with deletion at beg, end, sp. position.

void del_beg(node* head)
{
    if(head->next==NULL)
        return;
    else
        head = head->next;
    free(head);
}
void del_end(node* head)
{
    node *temp=head; if(head==NULL) return;
    while(temp->next->next!=NULL)

    if(head->next==NULL)
        head==NULL;

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

    temp->next=NULL;
    free(head);
}
```

```
void del_sp(node* head, int pos)
{
    bool notfound=false; node* temp=head;
    for(int i=1; i<pos; i++)
    {
        if(temp->next==NULL)
        {
            notfound=true;
            break;
        }
        temp=temp->next;
    }
    if(notfound)
        printf("Not Found\n");
    else
    {
        temp=temp->next->next;
    }
    free(head);
}
```



Code:

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

typedef struct node node;
struct node {
    int value;
    struct node *next;
};

node* createNode(int x) {
    node *n = (node*)malloc(sizeof(node));
    n->value = x;
    n->next = NULL;
    return n;
}

void inst_beg(node* head, int value) {
    node *new = createNode(value);
    new->next = head->next;
```

```
    head->next = new;
    printf("Inserted %d at begin\n", value);
}

void inst_end(node* head, int value) {
    node *new = createNode(value);
    node *temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new;
    printf("Inserted %d at end\n", value);
}

void display(node* head) {
    node *temp = head->next;
    if (temp == NULL) {
        printf("No element\n");
        return;
    }
    while (temp != NULL) {
        printf("%d\t", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

void del_beg(node* head) {
    if (head == NULL || head->next == NULL) {
        return;
    }
    node* temp = head->next;
    head->next = temp->next;
    free(temp);
    printf("Deleted from the beginning\n");
}

void del_end(node* head) {
    if (head == NULL || head->next == NULL) {
        return;
    }
    node* temp = head;
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }
```

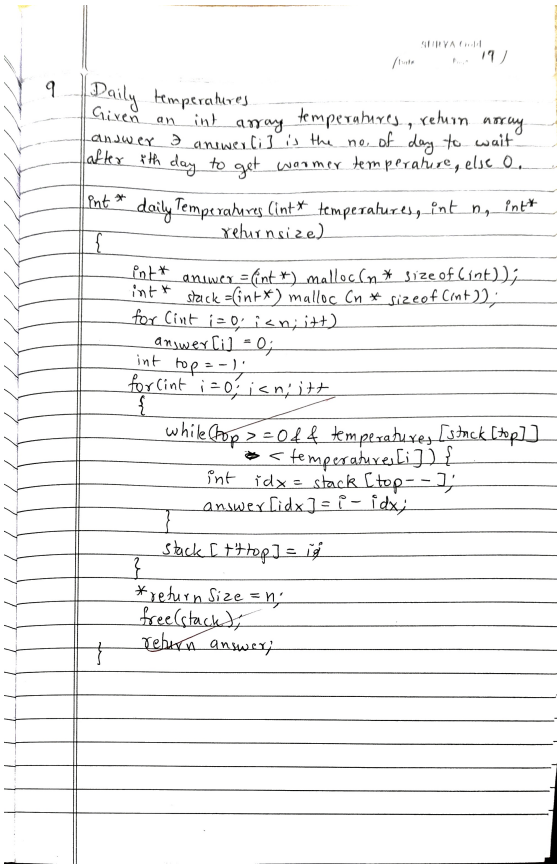
```
    node* toDelete = temp->next;
    temp->next = NULL;
    free(toDelete);
    printf("Deleted from the end\n");
}

void del_sp(node* head, int pos) {
    if (head == NULL || head->next == NULL) {
        printf("List is empty\n");
        return;
    }

    node* temp = head;
    for (int i = 1; i < pos; i++) {
        if (temp->next == NULL) {
            printf("Position not found\n");
            return;
        }
        temp = temp->next;
    }

    node* toDelete = temp->next;
    if (toDelete != NULL) {
        temp->next = toDelete->next;
        free(toDelete);
        printf("Deleted from position %d\n", pos);
    }
}

int main() {
    node *head = (node *)malloc(sizeof(node));
    head->next = NULL;

    printf("1. Inst-beg\n2. Inst-end\n3. Display\n4. Del-beg\n5. Del-end\n6. Del-sp\n0. Exit\n");

    int choice;
    char c;
    do {
        printf("\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int x;
                printf("Enter node value: ");
```

```
            scanf("%d", &x);
            inst_beg(head, x);
            break;
        }
        case 2: {
            int y;
            printf("Enter node value: ");
            scanf("%d", &y);
            inst_end(head, y);
            break;
        }
        case 3:
            display(head);
            break;
        case 4:
            del_beg(head);
            break;
        case 5:
            del_end(head);
            break;
        case 6: {
            int pos;
            printf("Enter position: ");
            scanf("%d", &pos);
            del_sp(head, pos);
            break;
        }
        case 0:
            break;
        default:
            printf("Invalid choice\n");
    }

    if ((c = getchar()) == '\n' || c == EOF) {}

  } while (choice != 0);

  printf("\nExiting...\n");
  return 0;
}
```

OUTPUT

```
❯ gcc linkedList.c -o linkedList
❯ ./linkedList
1. Inst-beg
2. Inst-end
3. Display
4. Del-beg
5. Del-end
6. Del-sp
0. Exit

Enter choice: 1
Enter node value: 5
Inserted 5 at begin

Enter choice: 2
Enter node value: 6
Inserted 6 at end

Enter choice: 2
Enter node value: 7
Inserted 7 at end

Enter choice: 2
Enter node value: 8
Inserted 8 at end

Enter choice: 3
5       6       7       8

Enter choice: 3
5       6       7       8

Enter choice: 4
Deleted from the beginning

Enter choice: 3
5       6       7       8

Enter choice: 4
Deleted from the beginning

Enter choice: 3
6       7       8

Enter choice: 5
Deleted from the end

Enter choice: 6
Enter position: 2
Deleted from position 2

Enter choice: 3
6

Enter choice: 0

Exiting...
```

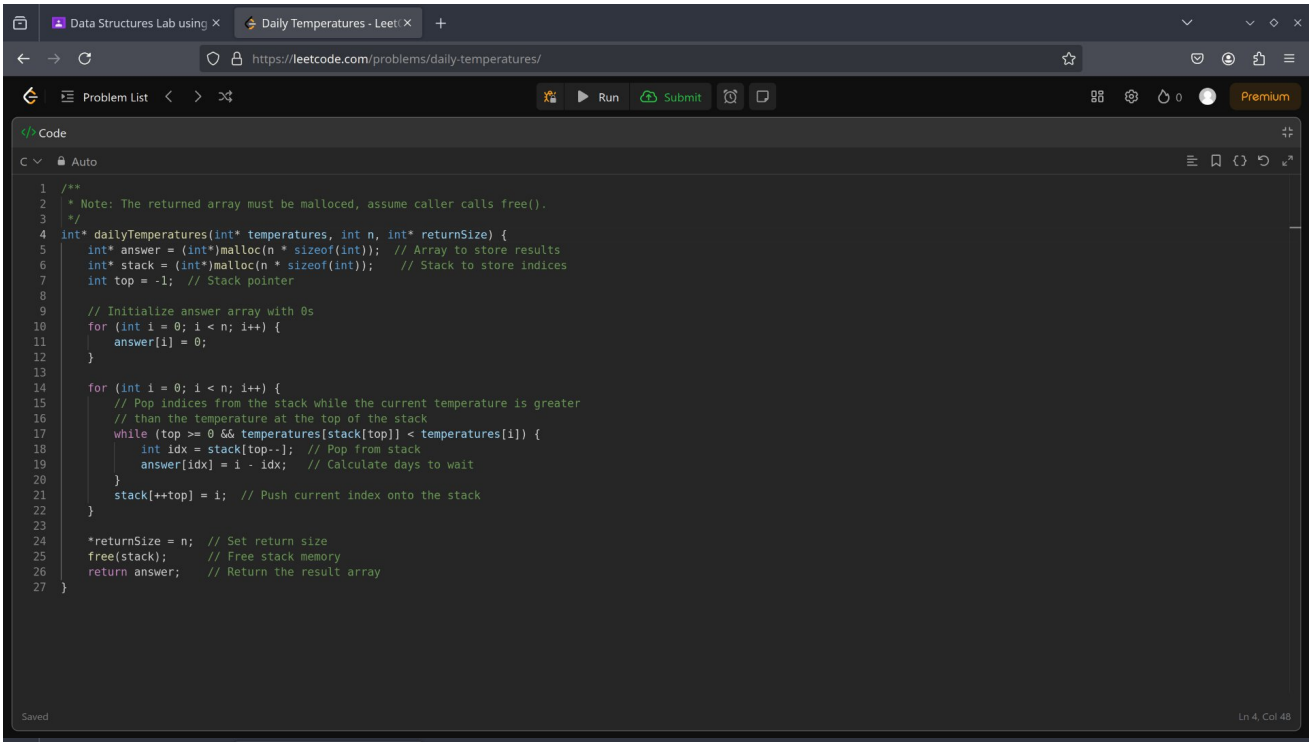
Program – LeetCode Question – Daily Temperatures

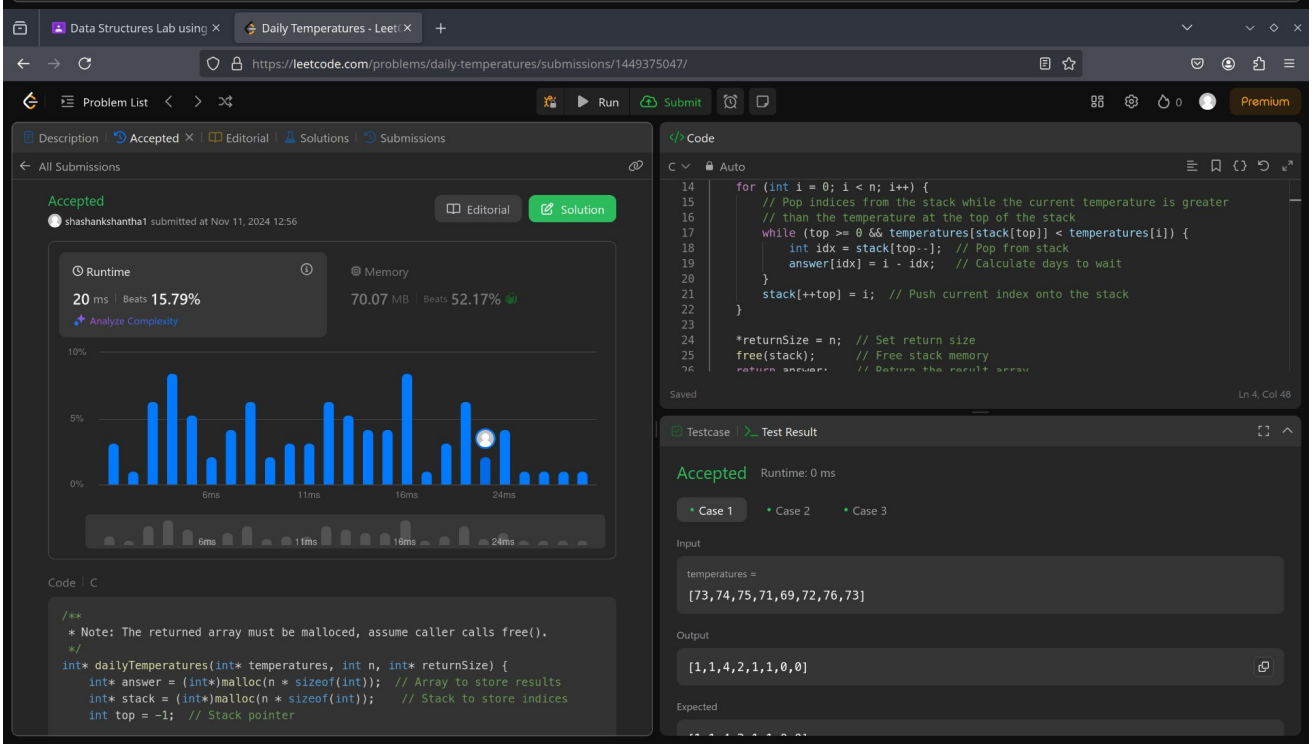


9 Daily temperatures
Given an int array temperatures, return array answer s answer[i] is the no. of day to wait after ith day to get warmer temperature, else 0.

```
int* dailyTemperatures(int* temperatures, int n, int*
        returnSize)
{
    int* answer = (int*) malloc(n * sizeof(int));
    int* stack = (int*) malloc (n * sizeof(int));
    for (int i=0; i<n; i++)
      answer[i] = 0;
    int top = -1;
    for(int i=0; i<n; i++
    {
      while(top >= 0 && temperatures[stack[top]]
            < temperatures[i]) {
        int idx = stack[top--];
        answer[idx] = i - idx;
      }
      stack[++top] = i;
    }
    *returnSize = n;
    free(stack);
    return answer;
}
```

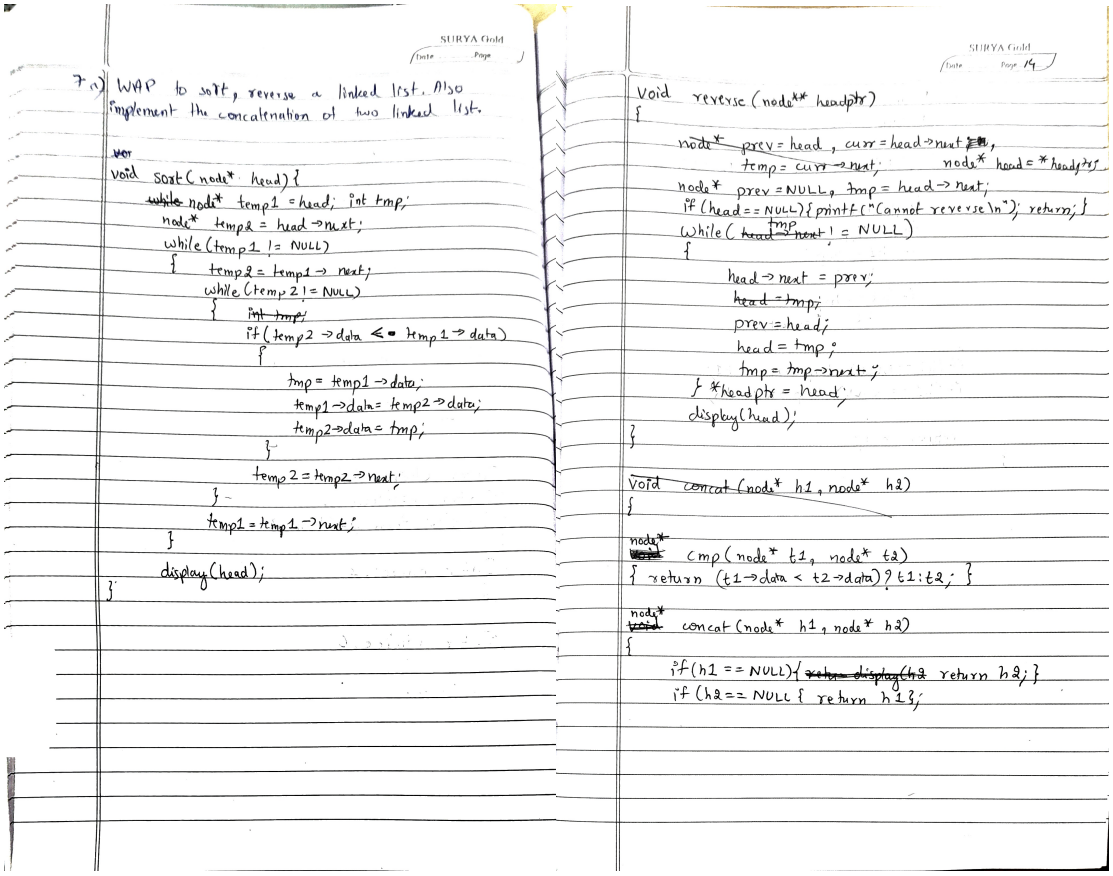
OUTPUT


Data Structures Lab using
Daily Temperatures - Leet
https://leetcode.com/problems/daily-temperatures/
Problem List
Run
Submit
Premium
Code
Auto
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* dailyTemperatures(int* temperatures, int n, int* returnSize) {
int* answer = (int*)malloc(n * sizeof(int)); // Array to store results
int* stack = (int*)malloc(n * sizeof(int)); // Stack to store indices
int top = -1; // Stack pointer
// Initialize answer array with 0s
for (int i = 0; i < n; i++) {
answer[i] = 0;
}
for (int i = 0; i < n; i++) {
// Pop indices from the stack while the current temperature is greater
// than the temperature at the top of the stack
while (top >= 0 && temperatures[stack[top]] < temperatures[i]) {
int idx = stack[top--]; // Pop from stack
answer[idx] = i - idx; // Calculate days to wait
}
stack[++top] = i; // Push current index onto the stack
}
*returnSize = n; // Set return size
free(stack); // Free stack memory
return answer; // Return the result array
}
Saved
Ln 4, Col 48


Data Structures Lab using
Daily Temperatures - Leet
https://leetcode.com/problems/daily-temperatures/submissions/1449375047/
Problem List
Run
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted
shashankshantha1 submitted at Nov 11, 2024 12:56
Editorial
Solution
Runtime
20 ms | Beats 15.79%
Analyze Complexity
Memory
70.07 MB | Beats 52.17%
10%
5%
0%
6ms
11ms
16ms
24ms
Code | C
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* dailyTemperatures(int* temperatures, int n, int* returnSize) {
int* answer = (int*)malloc(n * sizeof(int)); // Array to store results
int* stack = (int*)malloc(n * sizeof(int)); // Stack to store indices
int top = -1; // Stack pointer
Code
Auto
for (int i = 0; i < n; i++) {
// Pop indices from the stack while the current temperature is greater
// than the temperature at the top of the stack
while (top >= 0 && temperatures[stack[top]] < temperatures[i]) {
int idx = stack[top--]; // Pop from stack
answer[idx] = i - idx; // Calculate days to wait
}
stack[++top] = i; // Push current index onto the stack
}
*returnSize = n; // Set return size
free(stack); // Free stack memory
Saved
Ln 4, Col 48
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
temperatures =
[73,74,75,71,69,72,76,73]
Output
[1,1,4,2,1,1,0,0]
Expected

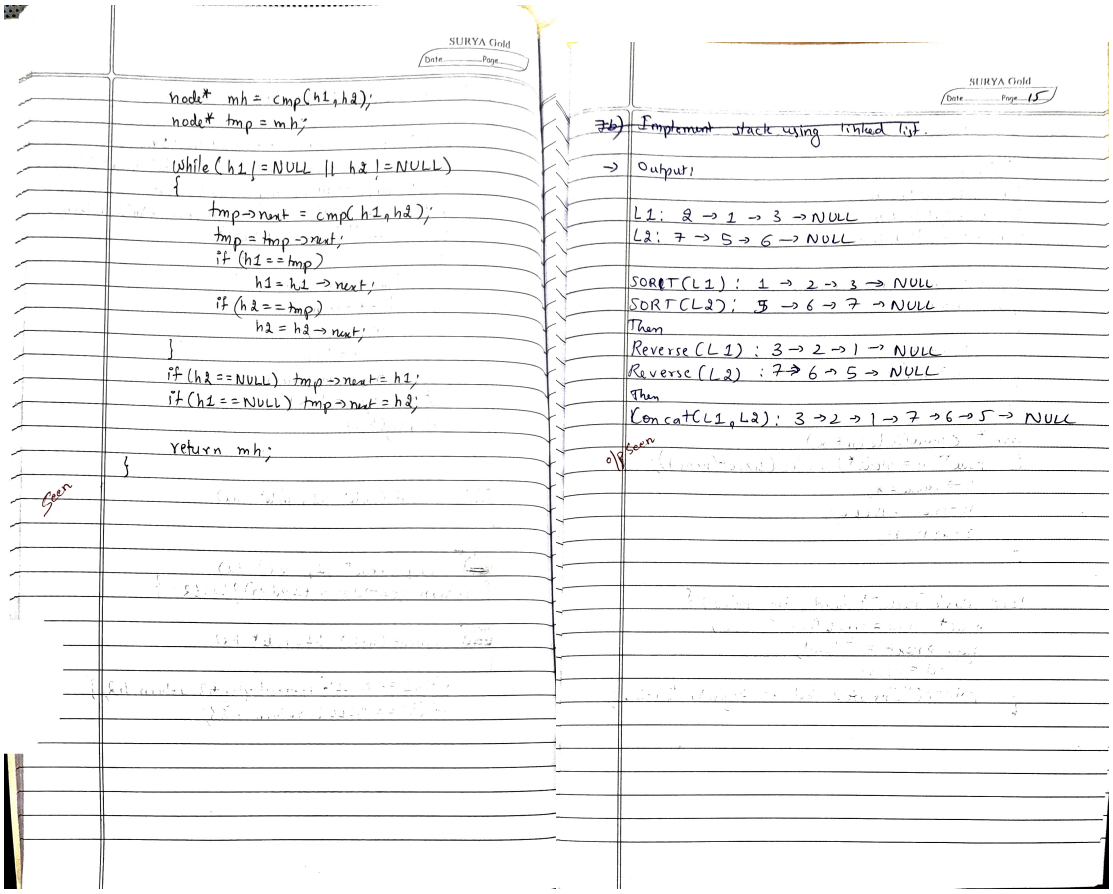
Program 6:

6a) WAP to Implement Single Link List with following operations:
Sort the linked list,
Reverse the linked list,
Concatenation of two linked lists.



```
7a) WAP to sort, reverse a linked list. Also
implement the concatenation of two linked list.

void sort(node* head){
  node* temp1 = head; int tmp;
  node* temp2 = head->next;
  while(temp1 != NULL)
  {
    temp2 = temp1->next;
    while(temp2 != NULL)
    {
      if(temp2->data <= temp1->data)
      {
        tmp = temp1->data;
        temp1->data = temp2->data;
        temp2->data = tmp;
      }
      temp2 = temp2->next;
    }
    temp1 = temp1->next;
  }
  display(head);
}
```

```
void reverse(node** headptr)
{
  node* prev = head, curr = head->next,
        temp = curr->next;   node* head = *headptr
  node* prev = NULL, tmp = head->next;
  if(head == NULL){printf("cannot reverse\n"); return;}
  while(tmp != NULL)
  {
    head->next = prev;
    head = tmp;
    prev = head;
    head = tmp;
    tmp = tmp->next;
  } *headptr = head;
  display(head);
}

void concat(node* h1, node* h2)
{

node*
  cmp(node* t1, node* t2)
{ return (t1->data < t2->data)? t1:t2; }

node*
  concat(node* h1, node* h2)
{
  if(h1 == NULL){ return h2; }
  if(h2 == NULL { return h1;
```



Code:

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

typedef struct node node;
struct node {
    int value;
    node* next;
};

node* createNode(int x) {
    node* n = (node*)malloc(sizeof(node));
    n->value = x;
    n->next = NULL;
    return n;
}

void inst_beg(node** head, int value) {
    node *new = createNode(value);
```

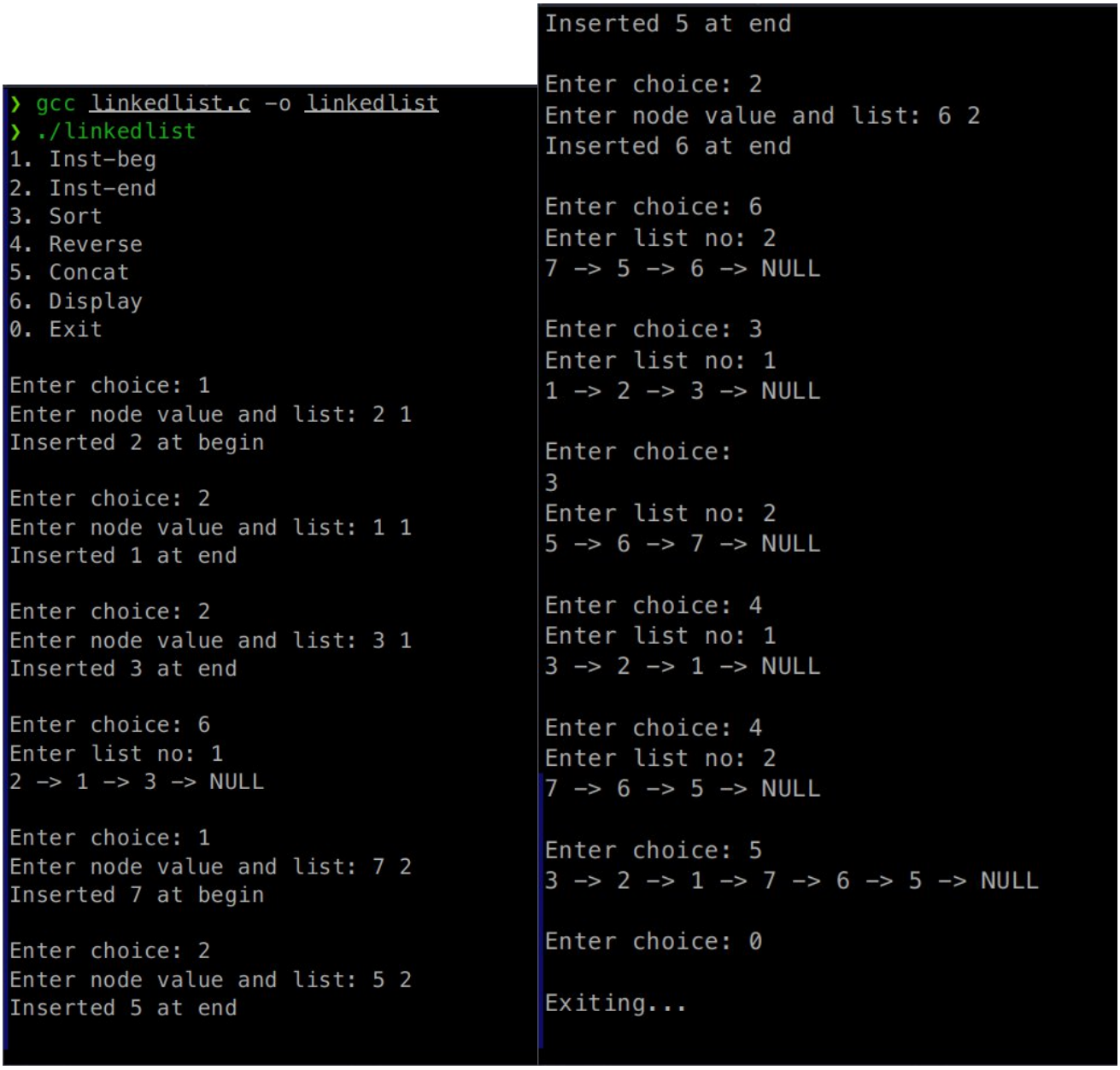
```
    new->next = *head;
    *head = new;
    printf("Inserted %d at begin\n", value);
}

void inst_end(node* head, int value) {
    node* new = createNode(value);
    node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = new;
    printf("Inserted %d at end\n", value);
}

void display(node* head) {
    node *temp = head;
    if (temp == NULL) {
        printf("No element\n");
        return;
    }
    while (temp != NULL) {
        printf("%d -> ", temp->value);
        temp = temp->next;
    }
    printf("NULL\n");
}


void sort(node* head)
{
        if (head == NULL) return;
        node* temp1 = head;
        while (temp1 != NULL)
        {
                node* temp2 = temp1->next;
                while (temp2 != NULL)
                {
                        if (temp2->value < temp1->value)
                        {
                                int tmp = temp1->value;
                                temp1->value = temp2->value;
                                temp2->value = tmp;
                        }
                        temp2 = temp2->next;
```

```
            }
            temp1 = temp1->next;
        }
        display(head);
}


void reverse(node** headptr)
{
        node* head = *headptr;
        node* prev = NULL;
        node* tmp = head ? head->next : NULL;

        while (tmp != NULL)
        {
            head->next = prev;
            prev = head;
            head = tmp;
            tmp = tmp->next;
        }
        head->next = prev; // Don't forget to point the new tail to NULL
        *headptr = head;
        display(head);
}


node* concat(node* h1, node* h2) {
    if (h1 == NULL) return h2;
    if (h2 == NULL) return h1;

    node* mh = NULL;
    node* tmp = NULL;

    // Initializing the head of the merged list by comparing the first nodes
    if (h1->value < h2->value) {
        mh = h1;
        h1 = h1->next;
    } else {
        mh = h2;
        h2 = h2->next;
    }

    tmp = mh; // set tmp to the head of the merged list

    // Merging the two lists
```

```
    while (h1 != NULL && h2 != NULL) {
        if (h1->value < h2->value) {
            tmp->next = h1;
            h1 = h1->next;
        } else {
            tmp->next = h2;
            h2 = h2->next;
        }
        tmp = tmp->next; // Move tmp to the last node
    }

    // If one of the lists is not empty, append the rest of the remaining list
    if (h1 != NULL) {
        tmp->next = h1;
    } else {
        tmp->next = h2;
    }

    // Display the merged list
    display(mh);

    return mh;
}

int main() {
    node* h1 = NULL;
    node* h2 = NULL;
    printf("1. Inst-beg\n2. Inst-end\n3. Sort\n4. Reverse\n5. Concat\n6. Display\n0. Exit\n");

    int choice;
    char c;
    do {
        printf("\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int x,y;
                printf("Enter node value and list: ");
                scanf("%d%d", &x, &y);
                (y==1)?inst_beg(&h1, x):inst_beg(&h2,x);
                break;
            }
            case 2: {
                int x,y;
```

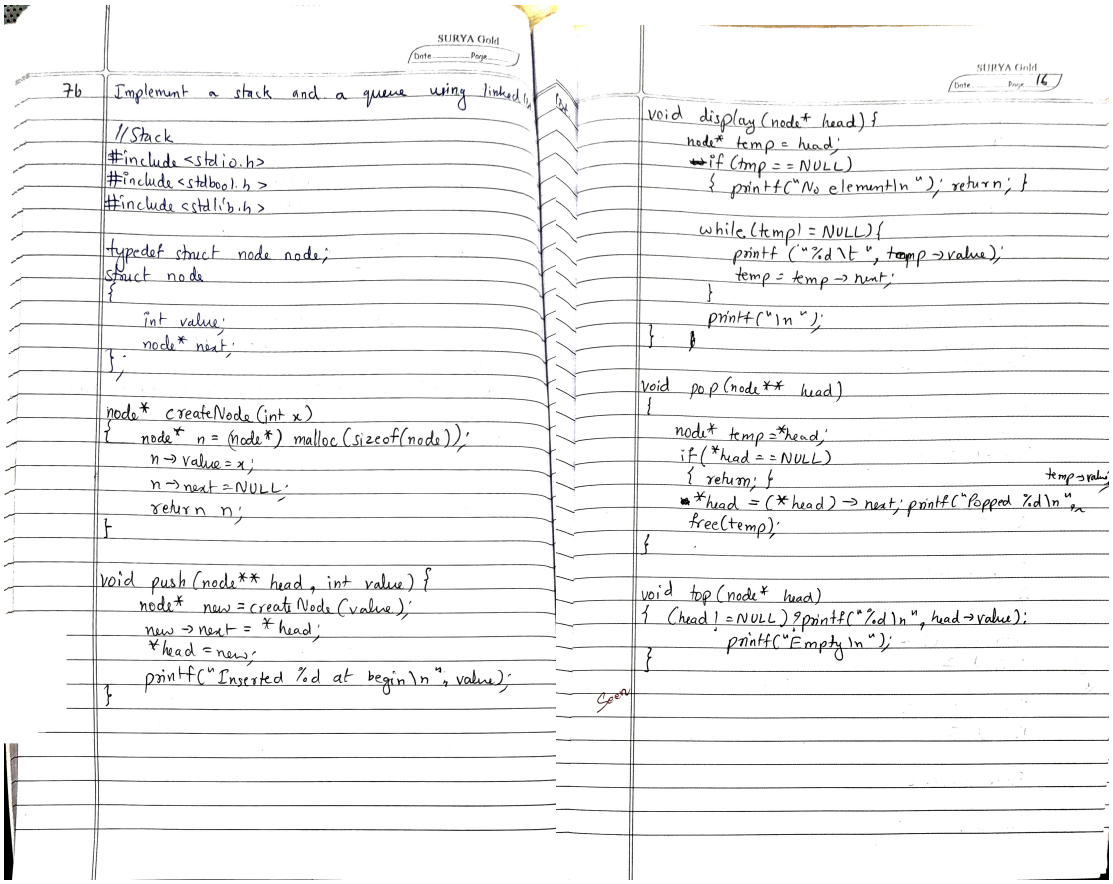
```
            printf("Enter node value and list: ");
            scanf("%d%d", &x, &y);
            (y==1)?inst_end(h1, x):inst_end(h2,x);
                break;
        }
        case 3:{
                int y;
            printf("Enter list no: ");
                scanf("%d",&y);
            sort((y==1)?h1:h2);
            break;
                }
        case 4:{
            int y;
            printf("Enter list no: ");
                scanf("%d",&y);
            reverse((y==1)?&h1:&h2);
            break;
                  }
        case 5:
            node* newList = concat(h1,h2);
            break;
        case 6: {
                int y;
            printf("Enter list no: ");
                scanf("%d", &y);
                display((y==1)?h1:h2);
                break;
                   }
        case 0:
            break;
        default:
            printf("Invalid choice\n");
      }

      if ((c = getchar()) == '\n' || c == EOF) {}

   } while (choice != 0);

   printf("\nExiting...\n");
   return 0;
}
```

OUTPUT



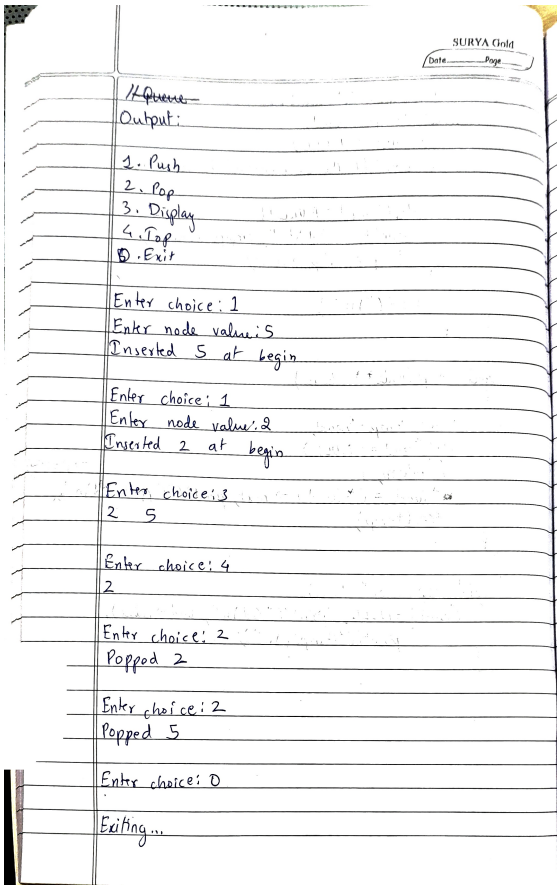
```
❯ gcc linkedlist.c -o linkedlist
❯ ./linkedlist
1. Inst-beg
2. Inst-end
3. Sort
4. Reverse
5. Concat
6. Display
0. Exit

Enter choice: 1
Enter node value and list: 2 1
Inserted 2 at begin

Enter choice: 2
Enter node value and list: 1 1
Inserted 1 at end

Enter choice: 2
Enter node value and list: 3 1
Inserted 3 at end

Enter choice: 6
Enter list no: 1
2 -> 1 -> 3 -> NULL

Enter choice: 1
Enter node value and list: 7 2
Inserted 7 at begin

Enter choice: 2
Enter node value and list: 5 2
Inserted 5 at end

Inserted 5 at end

Enter choice: 2
Enter node value and list: 6 2
Inserted 6 at end

Enter choice: 6
Enter list no: 2
7 -> 5 -> 6 -> NULL

Enter choice: 3
Enter list no: 1
1 -> 2 -> 3 -> NULL

Enter choice:
3
Enter list no: 2
5 -> 6 -> 7 -> NULL

Enter choice: 4
Enter list no: 1
3 -> 2 -> 1 -> NULL

Enter choice: 4
Enter list no: 2
7 -> 6 -> 5 -> NULL

Enter choice: 5
3 -> 2 -> 1 -> 7 -> 6 -> 5 -> NULL

Enter choice: 0

Exiting...
```

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



```
7b  Implement a stack and a queue using linked list

//Stack
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

typedef struct node node;
struct node
{
    int value;
    node* next;
};

node* createNode(int x)
{
    node* n = (node*) malloc(sizeof(node));
    n->value = x;
    n->next = NULL;
    return n;
}

void push(node** head, int value) {
    node* new = createNode(value);
    new->next = *head;
    *head = new;
    printf("Inserted %d at begin\n", value);
}
```

```
void display(node* head) {
    node* temp = head;
    if (temp == NULL)
    { printf("No element\n"); return; }

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

void pop(node** head)
{
    node* temp = *head;
    if(*head == NULL)
    { return; }
    *head = (*head)->next; printf("Popped %d\n", temp->value);
    free(temp);
}

void top(node* head)
{ (head != NULL) ? printf("%d\n", head->value);
        printf("Empty\n");
}
```



Code for Stack:

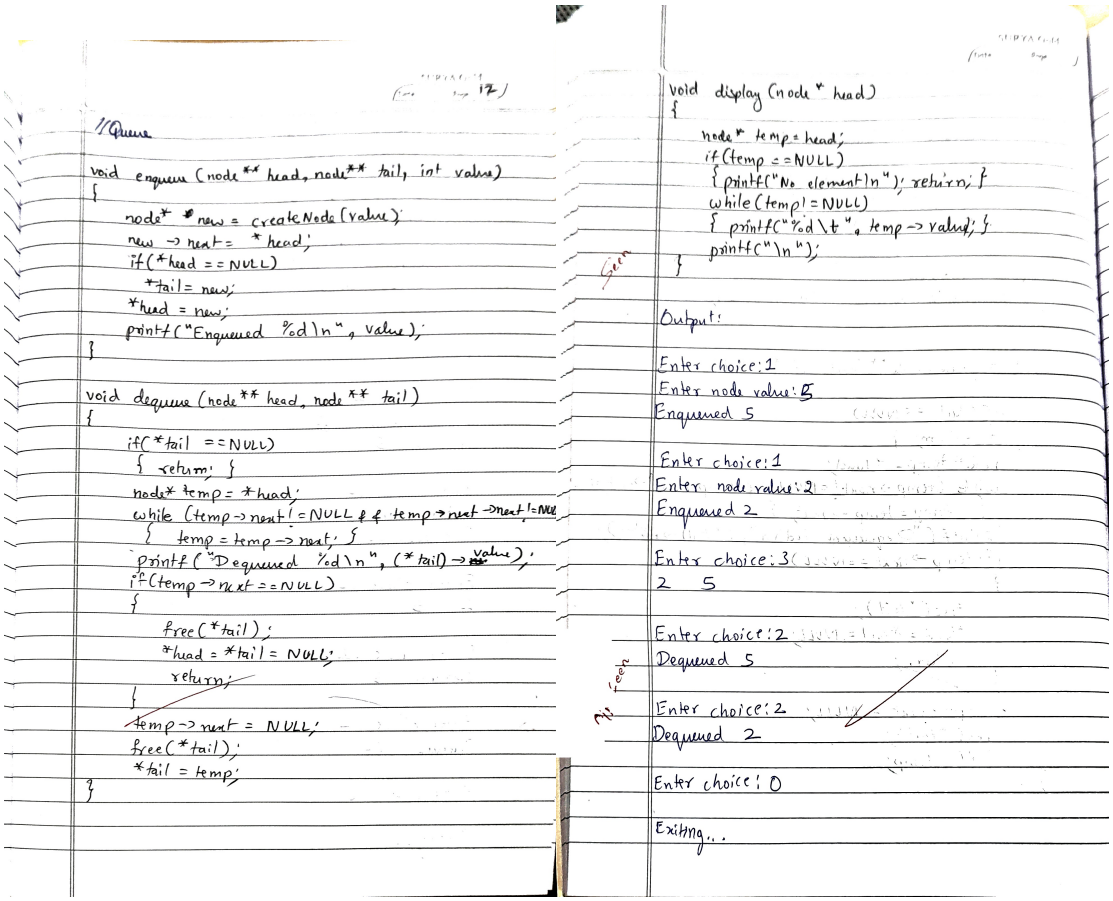
```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

typedef struct node node;
struct node {
    int value;
    node* next;
};

node* createNode(int x) {
    node *n = (node*)malloc(sizeof(node));
    n->value = x;
    n->next = NULL;
    return n;
}

void push(node** head, int value) {
    node *new = createNode(value);
```

```
    new->next = *head;
    *head = new;
    printf("Inserted %d at begin\n", value);
}



void display(node* head) {
    node* temp = head;
    if (temp == NULL) {
        printf("No element\n");
        return;
    }
    while (temp != NULL) {
        printf("%d\t", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

void pop(node** head) {
    if (*head == NULL) {
        return;
    }
    node* temp = *head;
    *head = (*head)->next;
    printf("Popped %d\n",temp->value);
    free(temp);
}

void top(node* head)
{
        (head!=NULL)?printf("%d\n",head->value):printf("Empty\n");
}

int main() {
    node* head = NULL;

    printf("1. Push\n2. Pop\n3. Display\n4. Top\n0. Exit\n");

    int choice;
    char c;
    do {
        printf("\nEnter choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1: {
                int x;
                printf("Enter node value: ");
                scanf("%d", &x);
                push(&head, x);
                break;
            }
            case 3:
                display(head);
                break;
            case 2:
                pop(&head);
                break;
                case 4: top(head);
                        break;
            case 0:
                break;
            default:
                printf("Invalid choice\n");
        }

        if ((c = getchar()) == '\n' || c == EOF) {}

    } while (choice != 0);

    printf("\nExiting...\n");
    return 0;
}
```

OUTPUT

```
❯ gcc stack.c -o stack
❯ ./stack
1. Push
2. Pop
3. Display
4. Top
0. Exit

Enter choice: 1
Enter node value: 2
Inserted 2 at begin

Enter choice: 1
Enter node value: 5
Inserted 5 at begin

Enter choice: 3
5       2

Enter choice: 4
5

Enter choice: 2
Popped 5

Enter choice: 2
Popped 2

Enter choice: 0

Exiting...
```



```
//Queue
void enqueue (node** head, node** tail, int value)
{
    node* new = createNode(value);
    new -> next = *head;
    if(*head == NULL)
      *tail = new;
    *head = new;
    printf("Enqueued %d\n", value);
}

void dequeue (node** head, node** tail)
{
    if(*tail == NULL)
    { return; }
    node* temp = *head;
    while (temp->next != NULL && temp->next->next != NULL)
      { temp = temp -> next; }
    printf("Dequeued %d\n", (*tail)->value);
    if(temp->next == NULL)
    {
      free(*tail);
      *head = *tail = NULL;
      return;
    }
    temp->next = NULL;
    free(*tail);
    *tail = temp;
}
```

```
void display (node* head)
{
    node* temp = head;
    if(temp == NULL)
    { printf("No element\n"); return; }
    while(temp != NULL)
    { printf("%d\t", temp->value); }
    printf("\n");
}
```

```
Output:

Enter choice: 1
Enter node value: 5
Enqueued 5

Enter choice: 1
Enter node value: 2
Enqueued 2

Enter choice: 3
2  5

Enter choice: 2
Dequeued 5

Enter choice: 2
Dequeued 2

Enter choice: 0

Exiting...
```

Code for Queue:

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

typedef struct node node;
struct node {
   int value;
   node* next;
};

node* createNode(int x) {
   node *n = (node*)malloc(sizeof(node));
   n->value = x;
   n->next = NULL;
   return n;
}

void enqueue(node** head, node** tail, int value) {
   node *new = createNode(value);
```

```
    new->next = *head;
    if(*head == NULL)
            *tail=new;
    *head = new;
    printf("Enqueued %d\n", value);
}



void display(node* head) {
    node* temp = head;
    if (temp == NULL) {
        printf("No element\n");
        return;
    }
    while (temp != NULL) {
        printf("%d\t", temp->value);
        temp = temp->next;
    }
    printf("\n");
}



void dequeue(node** head, node** tail) {
    if (*tail == NULL) {
        return;
    }
    node* temp = *head;
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }
    printf("Dequeued %d\n",(*tail)->value);
    if(temp->next==NULL)
    {
            free(*tail);
            *head = *tail = NULL;
            return;
    }
    temp->next = NULL;
    free(*tail);
    *tail = temp;
}
```

```
int main() {
    node* head = NULL;
    node* tail = NULL;
    printf("1. Enqueue\n2. Dequeue\n3. Display\n0. Exit\n");

    int choice;
    char c;
    do {
        printf("\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int x;
                printf("Enter node value: ");
                scanf("%d", &x);
                enqueue(&head, &tail, x);
                break;
            }
            case 2: {
                dequeue(&head, &tail);
                break;
            }
            case 3:
                display(head);
                break;
            case 0:
                break;
            default:
                printf("Invalid choice\n");
        }

        if ((c = getchar()) == '\n' || c == EOF) {}

    } while (choice != 0);

    printf("\nExiting...\n");
    return 0;
}
```

OUTPUT

```
❯ gcc queue.c -o queue
❯ ./queue
1. Enqueue
2. Dequeue
3. Display
0. Exit

Enter choice: 1
Enter node value: 2
Enqueued 2

Enter choice: 1
Enter node value: 5
Enqueued 5

Enter choice: 3
5       2

Enter choice: 2
Dequeued 2

Enter choice: 2
Dequeued 5

Enter choice: 2

Enter choice: 0

Exiting...
```

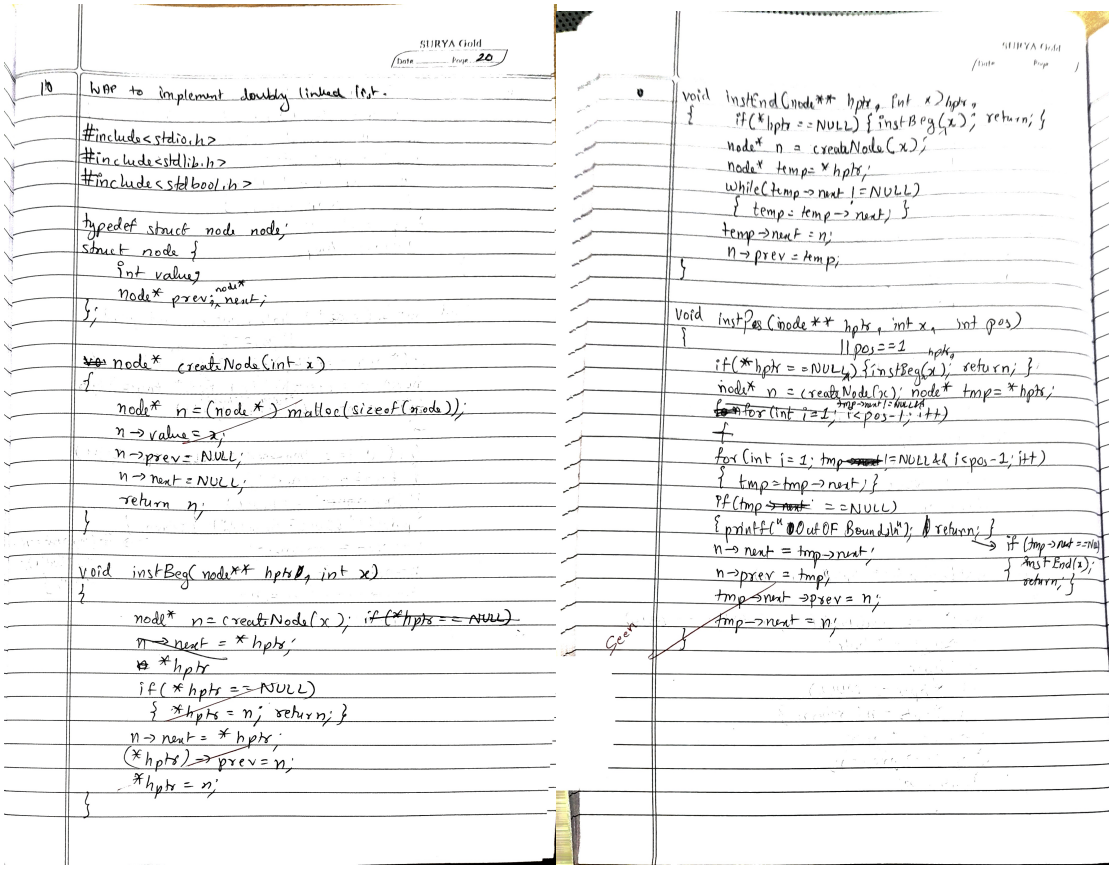
Program 7:

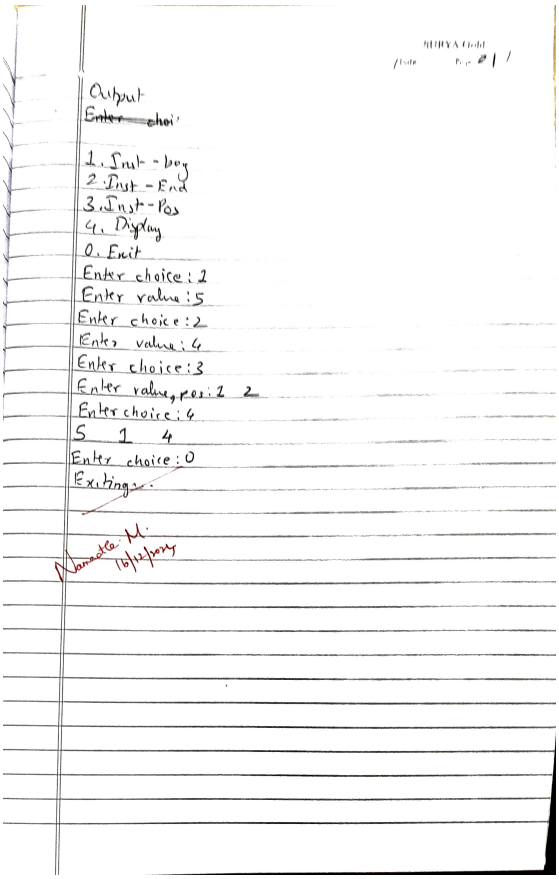
WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

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

c) Display the contents of the list





Code:

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

typedef struct node node;
struct node{
        int value;
        node* prev;node* next;
};

node* createNode(int x)
{
        node* n = (node*)malloc(sizeof(node));
        n->value=x;
        n->prev=NULL;
        n->next=NULL;
}
```

```
void instBeg(node** hptr, int x)
{
	node* n = createNode(x);
	if(*hptr == NULL)
	{
		*hptr=n; return;
	}
	n->next=*hptr;
	(*hptr)->prev=n;
	*hptr=n;
}

void instEnd(node** hptr, int x)
{
	if(*hptr==NULL){instBeg(hptr, x);return;}
	node* n = createNode(x);
	node* temp = *hptr;

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

	temp->next=n;
	n->prev=temp;
}

void instPos(node** hptr, int x, int pos)
{
	if(*hptr==NULL||pos==1){instBeg(hptr, x);return;}
	node* n = createNode(x);
	node* tmp=*hptr;

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

	if(tmp==NULL){printf("OutOfBounds\n"); return;}
	if(tmp->next==NULL){instEnd(hptr, x);return;}
	n->next=tmp->next;
	n->prev=tmp;
	tmp->next->prev=n;
	tmp->next=n;
}

void display(node* head)
{
	node* temp=head;
```

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

void main()
{
	node* head = NULL;
	int choice;
	printf("1.Inst-beg\n2.Inst-End\n3.Inst-Pos\n4.Display\n0.Exit\n");
	do{
		printf("Enter choice:");
		scanf("%d",&choice);
		int x;
		switch(choice)
		{
			case 1: printf("Enter value:");
				scanf("%d",&x);
				instBeg(&head, x);
				break;
			case 2: printf("Enter value:");
				scanf("%d",&x);
				instEnd(&head, x);
				break;
			case 3: int pos;
				printf("Enter value, pos:");
				scanf("%d%d",&x,&pos);
				instPos(&head, x, pos);
				break;
			case 4: display(head);
				break;
			case 0: break;
			default:printf("Invalid input\n");
		}
	}while(choice!=0);
	printf("Exiting...");
}
```

OUTPUT

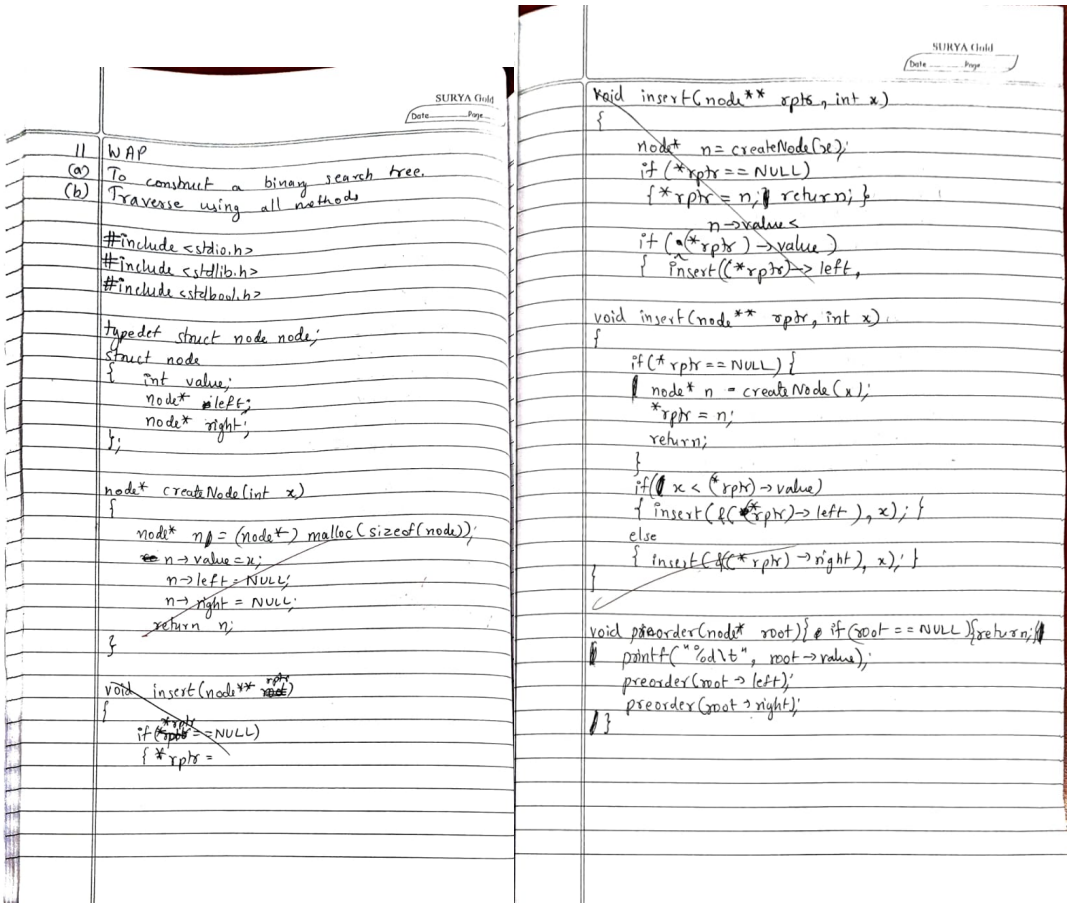
```
❯ gcc double.c -o double
❯ ./double
1.Inst-beg
2.Inst-End
3.Inst-Pos
4.Display
0.Exit
Enter choice:1
Enter value:5
Enter choice:2
Enter value:4
Enter choice:3
Enter value, pos:1 2
Enter choice:4
5       1       4
Enter choice:0
Exiting...%
```

Program 8 :

Write a program

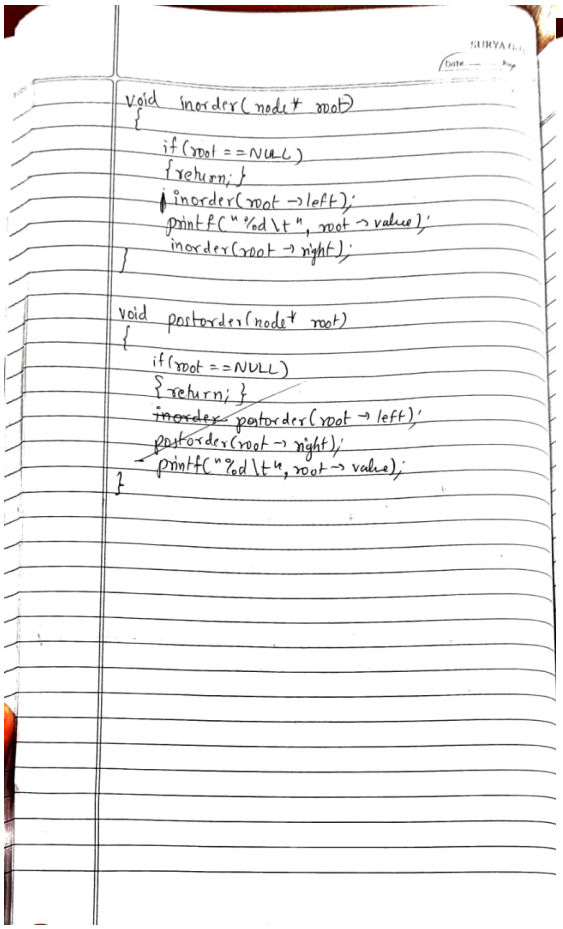
a) To construct a binary Search tree.

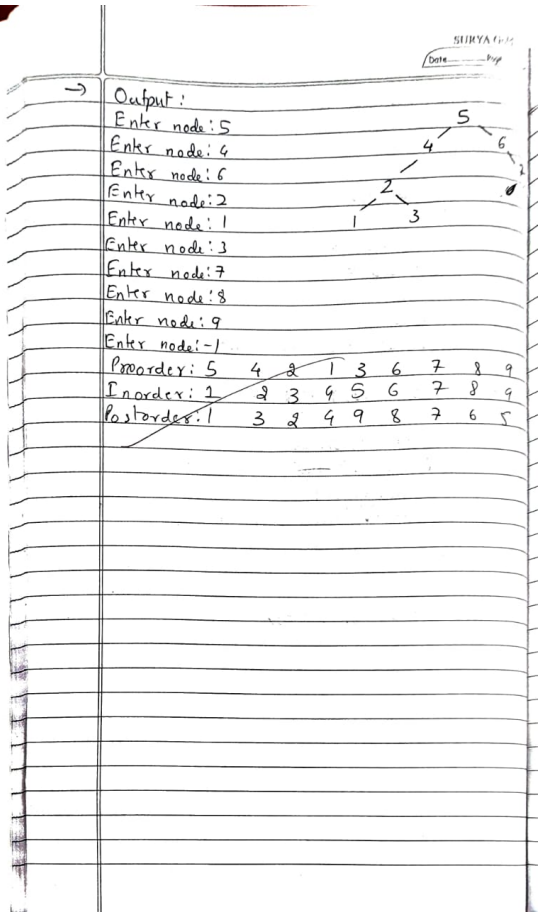
b) To traverse the tree using all the methods i.e., in-order, preorder and post order, display all traversal order



```
// WAP
// (a) To construct a binary search tree.
// (b) Traverse using all methods

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

typedef struct node node;
struct node
{
    int value;
    node* left;
    node* right;
};

node* createNode(int x)
{
    node* n = (node*) malloc(sizeof(node));
    n->value = x;
    n->left = NULL;
    n->right = NULL;
    return n;
}

void insert(node** rptr)
{
    if(*rptr == NULL)
    { *rptr =
```

```
void insert(node** rptr, int x)
{
    node* n = createNode(x);
    if (*rptr == NULL)
    {*rptr = n; return n;}
        n->value <
    if ((*rptr)->value)
    { insert((*rptr)->left,

void insert(node** rptr, int x)
{
    if(*rptr == NULL){
    node* n = createNode(x);
     *rptr = n;
     return;
    }
    if(x < (*rptr)->value)
    { insert(&((*rptr)->left), x); }
    else
    { insert(&((*rptr)->right), x); }
}

void preorder(node* root){ if(root == NULL){return;}
    printf("%d\t", root->value);
    preorder(root->left);
    preorder(root->right);
}
```





Code:

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

typedef struct node node;
struct node
{
	int value;
	node* left;
	node* right;
};

node* createNode(int x)
{
	node* n = (node*)malloc(sizeof(node));
	n->value=x;
	n->left=NULL;
	n->right=NULL;
	return n;
```

```
}

void insert(node** rptr, int x)
{
	if(*rptr==NULL)
	{
		node* n = createNode(x);
		*rptr=n;
		return;
	}
	if(x<(*rptr)->value)
	{insert(&((*rptr)->left), x);}
	else
	{insert(&((*rptr)->right), x);}
}

void preorder(node* root)
{
	if(root==NULL){return;}
	printf("%d\t", root->value);
	preorder(root->left);
	preorder(root->right);
}

void inorder(node* root)
{
	if(root==NULL){return;}
	inorder(root->left);
	printf("%d\t", root->value);
	inorder(root->right);
}

void postorder(node* root)
{
	if(root==NULL){return;}
	postorder(root->left);
	postorder(root->right);
	printf("%d\t", root->value);
}

void main()
{
	node* root = NULL;
	int x=0;
	while(x!=-1)
```

```
        {
                printf("Enter node:");
                scanf("%d",&x);
                if(x!=-1){insert(&root, x);}
        }
        printf("Preorder:");
        preorder(root);
        printf("\n");
        printf("Inorder:");
        inorder(root);
        printf("\n");
        printf("Postorder:");
        postorder(root);
        printf("\n");
}
```

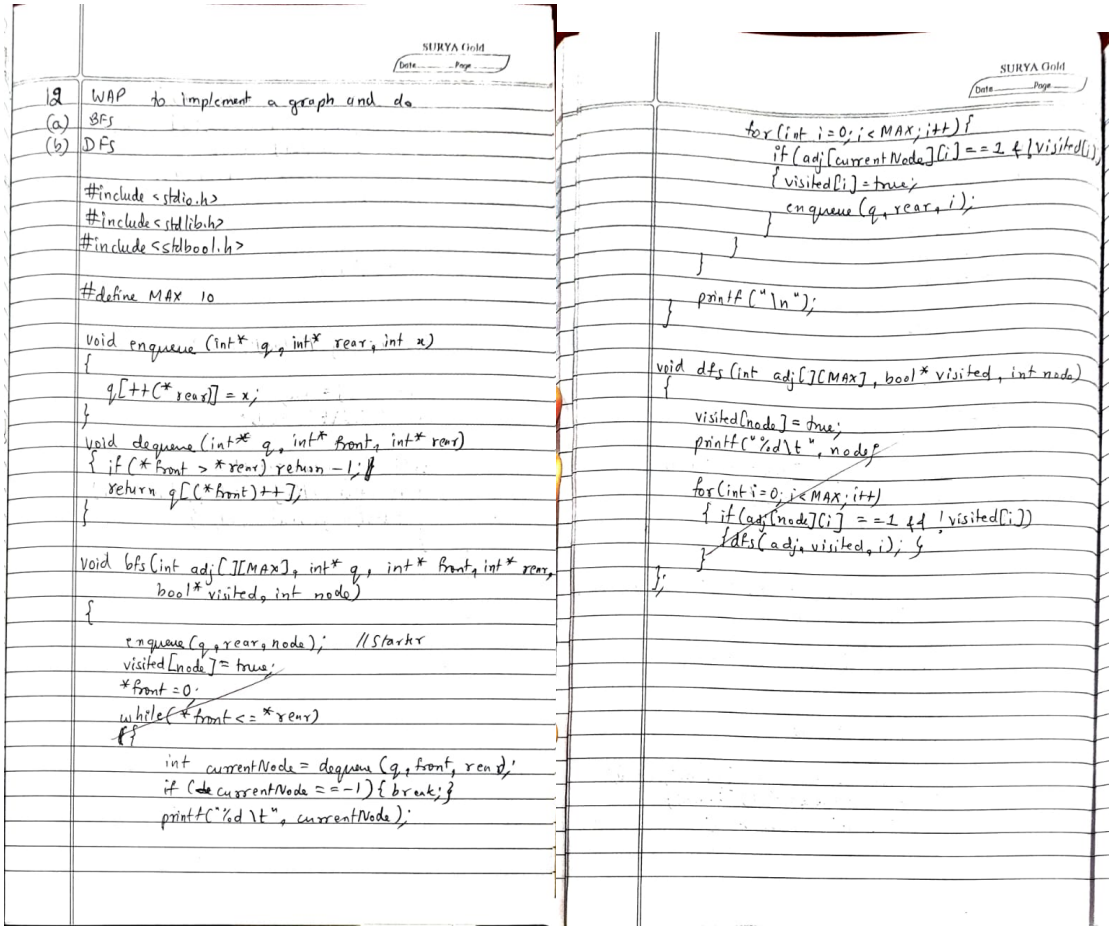
OUTPUT

```
❯ gcc tree.c -o tree
❯ ./tree
Enter node:5
Enter node:4
Enter node:6
Enter node:2
Enter node:1
Enter node:3
Enter node:7
Enter node:8
Enter node:9
Enter node:-1
Preorder:5      4       2       1       3       6       7       8       9
Inorder:1       2       3       4       5       6       7       8       9
Postorder:1     3       2       4       9       8       7       6       5
```

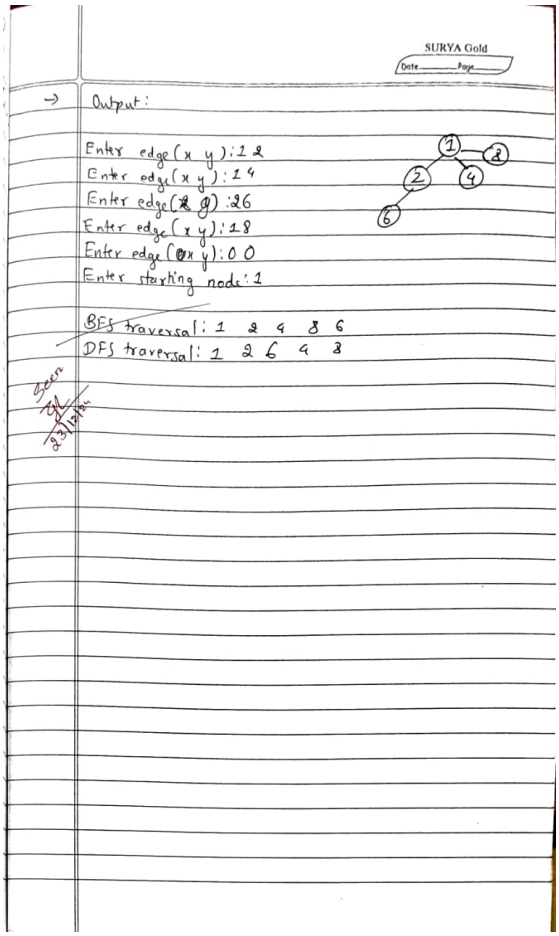
Program 9:

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

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



```
12  WAP to implement a graph and do
(a) BFS
(b) DFS

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

#define MAX 10

void enqueue (int* q, int* rear, int x)
{
   q[++(*rear)] = x;
}
void dequeue (int* q, int* front, int* rear)
{ if (*front > *rear) return -1;
  return q[(*front)++];
}

void bfs (int adj[][MAX], int* q, int* front, int* rear,
          bool* visited, int node)
{
    enqueue (q, rear, node);   //Starter
    visited[node] = true;
    *front = 0;
    while (*front <= *rear)
    {
        int currentNode = dequeue (q, front, rear);
        if (currentNode == -1) { break; }
        printf ("%d \t", currentNode);
        for (int i = 0; i < MAX; i++) {
            if (adj[currentNode][i] == 1 && !visited[i])
            { visited[i] = true;
              enqueue (q, rear, i);
            }
        }
    }
    printf ("\n");
}

void dfs (int adj[][MAX], bool* visited, int node)
{
    visited[node] = true;
    printf ("%d \t", node);

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



Code:

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

#define MAX 10

// Function to enqueue an element into the queue
void enqueue(int* q, int* rear, int x) {
    q[++(*rear)] = x;
}

// Function to dequeue an element from the queue
int dequeue(int* q, int* front, int* rear) {
    if (*front > *rear) return -1; // Queue is empty
    return q[(*front)++];
}

// Function to perform BFS
void bfs(int adj[][MAX], int* q, int* front, int* rear, bool* visited, int node) {
```

```
    enqueue(q, rear, node);
    visited[node] = true;
    *front = 0; // Set front to 0 after the first enqueue

    while (*front <= *rear) {
        int currentNode = dequeue(q, front, rear);

        if (currentNode == -1) {
            break;
        }

        printf("%d\t", currentNode); // Display the visited node

        // Check all adjacent nodes
        for (int i = 0; i < MAX; i++) {
            if (adj[currentNode][i] == 1 && !visited[i]) {
                visited[i] = true;
                enqueue(q, rear, i); // Enqueue the unvisited node
            }
        }
    }
    printf("\n");
}

// Function to perform DFS
void dfs(int adj[][MAX], bool* visited, int node) {
    visited[node] = true;
    printf("%d\t", node); // Display the visited node

    // Visit all adjacent unvisited nodes recursively
    for (int i = 0; i < MAX; i++) {
        if (adj[node][i] == 1 && !visited[i]) {
            dfs(adj, visited, i);
        }
    }
}

int main() {
    int adj[MAX][MAX] = {0}; // Adjacency matrix
    bool visited[MAX] = {false}; // Visited array
    int q[MAX]; // Queue for BFS
    int front = -1, rear = -1; // Queue pointers
    int x, y;

    // Input the graph as an undirected graph
```

```
    do {
        printf("Enter edge (x y): ");
        scanf("%d%d", &x, &y);
        if (x == y) {
            break;
        }
        adj[x][y] = 1;
        adj[y][x] = 1; // Since it's an undirected graph
    } while (x != y);

    printf("Enter starting node: ");
    scanf("%d", &x);

    // Perform BFS
    printf("BFS traversal: ");
    bfs(adj, q, &front, &rear, visited, x);

    // Reset visited array for DFS
    for (int i = 0; i < MAX; i++) {
        visited[i] = false;
    }

    // Perform DFS
    printf("DFS traversal: ");
    dfs(adj, visited, x);
    printf("\n");

    return 0;
}
```

OUTPUT

```
❯ gcc graph.c -o graph
❯ ./graph
Enter edge (x y): 1 2
Enter edge (x y): 1 4
Enter edge (x y): 2 6
Enter edge (x y): 1 8
Enter edge (x y): 0 0
Enter starting node: 1
BFS traversal: 1        2       4       8       6
DFS traversal: 1        2       6       4       8
```