LAB-1

Date 26/02/2024
Page 01

(1) write a program to push, pop and display the elements of a stack.

I/P

```
#include <stdio.h>
void push();
void pop();
void display();
int stack[50], i, j, n, option=0, top=-1;
void main(){
  printf("Enter the number of elements:");
  scanf("%d", &n);
  while(option!=4){
    printf("Choose one from the below option:\n");
    printf("\n1) Push\n2) Pop\n3) display\n4) Exit");
    scanf("%d", &option);
    switch(option){
      case 1:{
        push();
        break;
      }
      case 2:{
        pop();
        break;
      }
      case 3:{
        display();
        break;
      }
      case 4:{
        printf("Exit");
        break;
      }
      default:{
```

Date __/__/__
Page 02

```
        printf("Select the proper option.");
    }
  }
 }
}

void push(){
  int value;
  if(top==n-1){
    printf("\n OVERFLOW");
  }
  else{
    printf("Enter the value:");
    scanf("%d",&value);
    top=top+1;
    stack[top]=value;
  }
}

void pop(){
  if(top==-1){
    printf("\n UNDERFLOW");
  }
  else{
    printf("The poped element will be %d \n",
           stack[top]);
    top=top-1;
  }
}

void display(){
  for(i=top;i>=0;i--){
    printf("%d \n",stack[i]);
```

```
    }
    if(top==-1){
        printf("stack is empty.");
    }
}
```

O/P Enter the number of elements: 5
choose one from the below option:
1> push
2> pop
3> display
4> Exit
1
Enter the value: 8
choose one from the below option:
1> push
2> pop
3> display
4> Exit
6
select the proper option.
choose one from the below option:
1> push
2> pop
3> display
4> Exit
3
8
choose one from the below option:
1> push
2> pop
3> display
4> Exit

2
The poped element will be 8 ;
Choose one from the below option:
1) Push
2) Pop
3) display
4) Exit
4 :
Expt

26.12.24 (2) write a program to convert an infix expression into its equivalent postfix notation

```
I/P  char infix[20], post[20], stack[20];
     int top=-1, char result; int p=0;
     void push(char);
     char pop();
     int order(char);
     void main(){
       int i=0;
       printf("Enter infix:");
       int l=strlen(infix);
       while(i<l){
         symb=infix[i];
         switch(symb){
           case '(':
             push(symb);
             break;
           case ')':
             while(stack[top]!='('){
               result=pop();
               post[p++]=result;
             }
```

```
        top--;
        break;
    case '+':
    case '-':
    case '^':
    case '*':
    case '/':
        while (order(symb) < order(stack[top])){
            result = pop();
            post[p++] = result;
        }
        if (order(symb) > order(stack[top])) {
            push(symb)
        }
        else if (order(symb) == order(stack[top])
            && (order(symb) == 1) || (order(symb) == 2))
        {
            result = pop();
            post[p++] = result;
            push(symb);
        }
        else if(order(symb) == order(stack[top]) &&
            order(symb) == 3)
        {
            push(symb)
        }
        break;
    }
    i++;
}
while(top) >= 0) {
    post[p++] = stack[top];
}
```

```
void push(char x)
{
    top++;
    stack[top]=x;
}

char pop(){
    char re=stack[top];
    top--;
    return re;
}

int order(char symb)
{ int p;
switch(symb)
    {
        case'+':
        case'-':
            p=1;
        case'A':
            p=3;
        case'*':
        case'/':
            p=2;
    }
}
```

OIP Enter infix: A*B+C*D-E
Postfix is: AB*CD*+E-

LAB-2

Date __/__/__
Page _____

Q-1 Write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display.
The program should print appropriate message for overflow and underflow condition.

I/P
```
#include<stdio.h>
#define n=2;
int q[n], rear=-1, front=-1;
void add(int);
void del();
void display();
void main(){
    int choice=0, num;
    while(choice!=4){
        printf("Enter choice:");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Add number:");
                scanf("%d", &num);
                add(num);
                break;
            case 2:
                printf("Delete");
                del();
                break;
            case 3:
                printf("Display");
                display();
                break;
            case 4:
                printf("Exit");
```

```
            break;
        }
    }
}
void add(int p)
{
    if (front==-1 && rear==-1)
    {
        rear++;
        q[rear] = p;
    }
    else if (rear==n)
    {
        printf("Queue is FULL");
    }
    else
    {
        rear++;
        q[rear] = p;
    }
}
void del()
{
    if(rear==-1 && front==-1)
    {
        printf("Queue is empty");
    }
    else if((front+1) == (rear+1))
    {
        printf("Queue is empty now");
        front=-1, rear=-1;
    }
    else{
```

```
        front++;
        k = q[front];
        printf("%d is removed", k);
    }
}
void display() {
    int i;
    if (rear == -1 && front == -1) {
        printf("Queue is empty");
    }
    else {
        for (i = front+1; i <= rear; i++) {
            printf("%d", q[i]);
        }
        printf("\n");
    }
}
```

O/P

```
Enter choice: 1
Add number: 25
Enter choice: 1
Add number: 24
Enter choice: 3
display
25 24
Enter choice: 2
Delete
Enter choice: 2
Delete
Enter choice: 2
Queue is empty now
Enter choice: 4
Exit.
```

Date __/__/__
Page

Q2 Write a program to simulate the working of a circular queue using an array. Provide the following operations: Insert, delete & display. The program should print appropriate message for queue empty and queue overflow condition.

I/P
```
#include <stdio.h>
#define n 3
void enqueue(int); int q[n];
void dequeue();
void display();
int rear=-1, front=-1;
void main(){
    enqueue(25);
    enqueue(34);
    enqueue(45);
    enqueue(24);
    display();
    dequeue();
    dequeue();
    enqueue(44);
    display();
}
void enqueue(int p){
    if(rear==-1 && front==-1)
    {
        rear++;
        q[rear]=p;
    }
    else if(rear==front)
    {
        printf("Queue is full");
    }
```

```
    else {
        rear = (rear+1) % n;
        q[rear] = p;
    }
}
void dequeue(){
    int t;
    if(rear==-1 && front==-1){
        printf("Queue is empty");
    }
    else if((front+1)%n == (rear+1)){
        printf("Queue is empty now");
        rear=-1, front=-1;
    }
    else{
        front=(front+1)%n;
        t=q[front];
        printf("%d removed", t);
    }
}
void display(){
    int i,j;
    if(rear==-1 && front==-1){
        printf("Queue is empty");
    }
    else if(rear==front){
        i=front+1;
        while(i!=rear){
            printf("%d", q[i]);
            i=(i+1)%n;
        }
        printf("%d", q[i]);
```

```
printf("\n");
}
else{
    i = front+1;
    while (i != rear){
        printf("%d", q[i]);
        i = (i+1) % n;
    }
    printf("%d", q[i]);
    printf("\n");
}
```

O/P Queue is full

25 34 45

45 44

LAB-3

Date __/__/__
Page _____

Q-1 write a program 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 linked list.

I/P

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node* next;
};
struct Node* createNode(int data){
    struct Node* newNode=(struct Node*)
        malloc(sizeof(struct Node));
    if(newNode==NULL){
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data=data;
    newNode->next=NULL;
    return newNode;
}
struct Node* createLinkedList(int values[], int size){
    struct Node* head=NULL;
    struct Node* tail=NULL;
    for(int i=0; i<size; i++){
        struct Node* newNode=createNode(values[i]);
        if(head==NULL){
            head=newNode;
            tail=newNode;
        }
```

```
        else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}
void insertFirst(struct node** head, int data) {
    struct node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}
void insertAtPosition(struct node **head, int data,
                      int position) {
    if(position == 0) {
        insertFirst(head, data);
        return;
    }
    struct node* new_node = createNode(data);
    struct node* current = *head;
    for(int i=0; i<position-1; ++i) {
        if(current == NULL) {
            printf("Invalid position\n");
            return;
        }
        current = current->next;
    }
    new_node->next = current->next;
    current->next = new_node;
}
void insertEnd(struct node** head, int data) {
    struct node* new_node = createNode(data);
    if(*head == NULL) {
```

```
        *head = new_node;
        return;
    }
    struct Node* current = *head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = new_node;
}
void display(struct Node* node) {
    while (node != NULL) {
        printf("%d ->", node->data);
        node = node->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* linkedList = createLinkedList();
    int data;
    printf("Enter data to insert at beginning: ");
    scanf("%d", &data);
    insertFirst(&linkedList, data);
    int position;
    printf("Enter data to insert at a specific
            position: ");
    scanf("%d", &data);
    printf("Enter position: ");
    scanf("%d", &position);
    insertAtPosition(&linkedList, data, position);
    printf("Enter data to insert at end: ");
    scanf("%d", &data);
    insertEnd(&linkedList, data);
    display(linkedList);
    return 0;
```

Date __/__/__
Page

O/P
Enter number of elements: 4
Enter the elements:
1 2 3 4
Enter data to insert at the beginning: 5
Enter data to insert at a specific position: 34
Enter the position: 2
Enter data to insert at the end: 66
5 -> 1 -> 34 -> 2 -> 3 -> 4 -> 66 -> NULL

Q-2 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 list, Display the contents of linked list.

```
I/P #include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)
        malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

Date __/__/__
Page _____

```
struct Node* create(linked list){
    struct Node* head = NULL;
    struct Node* tail = NULL;
    int size, data;
    printf("Enter the number of elements:");
    scanf("%d", &size);
    printf("Enter the elements:\n");
    for(int i=0; i<size; ++i){
        scanf("%d", &data);
        struct Node* newNode = createNode(data);
        if(head == NULL){
            head = newNode;
            tail = newNode;
        }
        else{
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}
void deleteFirst(struct Node** head){
    if(*head == NULL){
        printf("List is empty. Nothing to delete\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}
void deleteElement(struct Node** head, int key){
    if(*head == NULL){
        printf("List is empty nothing to delete\n");
```

```
        return;
    }
    struct node * current = *head;
    struct node * prev = NULL;
    while(current != NULL && current->data != key)
        prev = current;
        current = current->next;
    }
    if(current == NULL) {
        printf("Elements not found in list\n");
        return;
    }
    if (prev == NULL) {
        *head = current->next;
    }
    else {
        prev->next = current->next;
    }
    free(current);
}
void deleteLast(struct node **head){
    if (*head == NULL){
        printf("List is empty. Nothing to delete");
        return;
    }
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }
    struct node* current = *head;
    struct node* prev = NULL;
    while(current->next != NULL) {
```

```
        prev = current;
        current = current->next;
    }
    prev->next = NULL;
    free(current);
}
void display(struct Node *head){
    while(head != NULL){
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}
int main(){
    struct Node* linkedlist = createLinkedList();
    printf("LinkedList before deletion:\n");
    display(LinkedList);
    deleteFirst(&LinkedList);
    printf("LinkedList after deleting first element:\n");
    display(LinkedList);
    int key;
    printf("Enter the element to delete:");
    scanf("%d", &key);
    deleteElement(&LinkedList, key);
    printf("LinkedList after deleting specified element:\n");
    display(LinkedList);
    deleteLast(&LinkedList);
    printf("LinkedList after deleting last element:\n");
    display(LinkedList);
    return 0;
}
```

Date
Page

O/P

Enter number of elements: 4

Enter the elements:

1 2 3 4

Linked List before deletion:

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

Linked list after deleting first element:

2 -> 3 -> 4 -> NULL

~~Enter the after deleting first element~~

Enter the element to delete: 4

Linked List after deleting specified element:

2 -> 3 -> NULL

Linked List after deleting last element:

2 -> NULL

LAB-4

Date __/__/__
Page _____

Q-) Write a program to sort, reverse and concatenate a linear linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node * next;
};
void insert_at_begin(struct node* head, int data)
    {
        struct node * newnode = (struct node*) malloc(
                (sizeof(struct node));
        newnode -> data = data;
        new node -> next = head;
        head = new node;
}
void printlist(struct node* head){
        while(head != NULL){
            printf(" %d", head -> data);
            head = head -> next;
        }
        printf("\n");
}
void sortlist(struct node** head){
        struct node* current, * next node;
        int temp;
        current = *head;
        while (current != NULL){
            next node = current -> next;
            while (next node != NULL){
                if (current -> data) next node -> data){
                    temp = current -> data;
```

```
            current->data = nextnode->data;
            nextnode->data = temp;
        }
        nextnode = nextnode->next;
    }
    current = current->next;
  }
}
void reverselist(struct node** headRef){
    struct node *prev, *current, *nextnode;
    prev = NULL;
    current = *headRef;
    while(current != NULL){
        nextnode = current->next;
        current->next = prev;
        prev = current;
        current = nextnode;
    }
    *headRef = prev;
}
void concatenateLists(struct node** list1, struct node** list2)
{
    if(*list1 == NULL){
        *list1 = list2;
        return;
    }
    struct node* temp = *list1;
    while(temp->next != NULL){
        temp = temp->next;
    }
    temp->next = list2;
}
int main(){
```

Date __/__/__
Page _____

```
struct node * list1 = NULL;
struct node * list2 = NULL;
int choice;
int data;
printf("1. Insert into list 1 \n");
printf("2. Insert into list 2 \n");
printf("3. Sort list 1 \n");
printf("4. Reverse list 1 \n");
printf("5. Concatenate lists \n");
printf("6. Print lists \n");
printf("0. Exit \n");
while(1){
    printf("Enter choice:");
    scanf("%d", &choice);
    switch(choice){
        case 1:
            printf("Enter data into list 1");
            scanf("%d", &data);
            insertatBeginning(&list1, data);
            break;
        case 2:
            printf("Enter data to insert into list 2:");
            scanf("%d", &data);
            insert_at_begin(&list2, data);
            break;
        case 3:
            sortList(&list1);
            printf("list 1 sorted");
            break;
        case 4:
            reverseList(&list1);
            printf("List 1 reversed \n");
            break;
```

```
        case 5:
            concatenateLists(&list1, list2);
            printf("Lists concatenated\n");
            break;
        case 6:
            printf("List 1\n");
            printList(list1);
            printf("List 2\n");
            printList(list2);
            break;
        case 7:
            printf("Exit"); exit(0);
            break;
    }
}
return 0;
}
```

O/P

1. Insert into list 1
2. Insert into list 2
3. Sort list 1
4. Reverse list 2.
5. concatenate lists
6. print lists
0. Exit.

Enter your choice: 1
Enter data to insert into list 1: 23
Enter your choice: 2
Enter data to insert into list 2: 34
Enter your choice: 1
Enter data to insert into list 1: 2
Enter data to insert into list 2: 45

Date __/__/__
Page _____

Enter your choice: 6
List 1: 2 23
List 2: 45 34
Enter your choice: 4
List 1 reversed
Enter your choice: 1
List 1: 23 2
List 2: 45 34
Enter your choice: 5
List concatenated
Enter your choice: 6
List 1: 23 2 45 34
List 2: 45 34.

Enter your choice: 0
Exit

Q-2 write a program to implement stack using linked list.

I/P
```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node * next;
};
struct node * top = NULL;
void push (int x) {
    struct node * newnode;
    newnode=(struct node *) malloc (sizeof (struct node));
    newnode => data = x;
    newnode -> next = top;
    top = newnode;
}
```

```
void pop() {
    struct node *temp;
    temp = top;
    if (top == 20) {
        printf("List is empty");
    }
    else {
        top = top->next;
        free(temp);
    }
}
void main() {
    int choice = -1; int x;
    while (choice != 4) {
        printf("Enter choice:");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Add at top:");
                scanf("%d", &x);
                push(x);
                break;
            case 2:
                printf("Pop from top");
                pop();
                break;
            case 3:
                printf("Display\n");
                display();
                break;
            case 4:
                printf("Exit");
                break;
        }
    }
```

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

O/P
```
Enter choice: 1
Add at top: 42
Enter choice: 1
Add at top: 43
Enter choice: 3
43 42
Enter choice: 2
Pop from top
Enter choice: 3
42
Enter choice: 4
Exit.
```

Q-3 Queue implementation using linked list.

I/P
```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node* next;
};
struct node* front = NULL, *rear = NULL;
void enqueue(int x) {
    newnode = (struct node*) malloc(sizeof(struct node));
```

```
    newnode->data=x; newnode->next=NULL;
    if(front==0 && rear==0){
        front=rear=newnode;
    }
    else{
        rear->next=newnode;
        rear=newnode;
    }
}
void display(){
    struct node *temp;
    if(front==0 && rear==0){
        printf("Queue is empty");
    }
    else{
        temp=front;
        while(temp!=NULL){
            printf("%d",temp->data);
            temp=temp->next;
        }
    }
}
void dequeue(){
    struct node *temp;
    temp=front;
    if(front==0 && rear==0){
        printf("Empty Queue");
    }
    else{
        front=front->next;
        free(temp);
    }
}
```

```
void main() {
    int x;
    int choice = -1;
    while (choice != 4) {
        printf("Enter choice:");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Add element:");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 2:
                printf("Delete element:");
                dequeue();
                break;
            case 3:
                printf("Display");
                display();
                break;
            case 4:
                printf("Exit");
                break;
        }
    }
}
```

O/P

```
Enter choice: 1
Add element: 12
Enter choice: 1
Add element: 13
Enter choice: 1
Add element: 15
```

Date __/__/__
Page ____

Enter choice: 3
Display
12 13 15
Enter choice: 2
Delete element
Enter choice: 3
Display
13 15
Enter choice: 4
Exit.

LAB-5

Date __/__/__
Page _____

Q-) write a program to implement doubly linked list with primitive operations
(a) Create a doubly linked list.
(b) Insert a new node to the left of node.
(c) Delete the node based on a specific value.

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *head=NULL, *newnode;
struct node *temp;
void create();
void insert_at_left();
void delete_at_value();
void display();
void create(int data){
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=data;
    newnode->next=NULL; newnode->prev=NULL;
    if(head==NULL){
        head=temp=newnode;
    }
    else{
        temp->next=newnode;
        newnode->prev=temp;
        temp=newnode;
    }
}
void insert_at_left(int data){
```

```
int i=0, pos; struct node* current = head;
newnode = (struct node*) malloc(sizeof(struct node));
newnode->next = NULL;
newnode->prev = NULL;
printf("Enter position:");
scanf("%d", &pos);
while(i < pos-1){
    current = current->next;
    i++;
}
newnode->next = current->next;
(current->next)->prev = newnode;
current->next = newnode;
newnode->prev = current;
}
void delete_at_value(){
    struct node* nextnode;
    int value; struct node* current = head;
    printf("Enter value to be deleted:");
    scanf("%d", &value);
    while(current->next != NULL){
        nextnode = current->next;
        if(nextnode->data != value){
            current = current->next;
            continue;
        }
        else{
            current->next = nextnode->next;
            (nextnode->next)->prev = current;
            free(nextnode);
            break;
        }
    }
}
```

Date __/__/__
Page _____

O/P

Enter choice: 1
Enter data for node: 5
Enter choice: 2
Enter data: 21
Enter position: 0
Enter choice: 1
Enter data for node: 6
Enter choice: 3
Enter value to be deleted: 5
Enter choice: 4
Display
21 6

LAB-6

Date __/__/__
Page _____

Q-1 Delete the middle node of a linked list -> Leet co

I/P

```
struct ListNode* deleteMiddle(struct ListNode* head){
    int n=0;
    struct ListNode *current=head, *nextnode, *prevnode;
    while(current!=NULL){
        current=current->next;
        n++;
    }
    int mid=n/2; int i=0;
    current=head;
    if(mid==0){
        free(current);
        head=NULL;
        return head;
    }
    else if(mid==1){
        prevnode=current;
        current=current->next;
        prevnode->next=current->next;
        free(current);
        return head;
    }
    else{
        while(i<mid-1){
            current=current->next;
            nextnode=current->next;
            i++;
        }
    current->next=nextnode->next;
    free(current);
    return head;
}
```

Date __/__/__
Page _____

Q-2 Odd Even Linked list -> Leet code:

I/P
```
struct ListNode* oddEvenList(struct ListNode* head){
    if(!head || !head->next || !head->next->next){
        return head;
    }
    struct ListNode *oddHead=head, *evenHead=head->next;
    struct ListNode *odd=oddHead, *even=evenHead;
    while(even && even->next){
        odd->next=even->next;
        odd=odd->next;
        even->next=odd->next;
        even=even->next;
    }
    odd->next=evenHead;
    return oddHead;
}
```

Q-3 write a program
(a) to construct Binary Search tree
(b) Transverse the tree using inorder, post-order, preorder.
(c) Display the elements in tree.

I/P
```
#include<stdio.h>
#include<stdlib.h>
struct TreeNode{
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};
struct TreeNode* createNode(int val){
```

Date __/__/__
Page _____

```
struct TreeNode* newNode = (struct TreeNode*)malloc(
                    sizeof(struct TreeNode));
newNode->val = val;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
struct TreeNode* insert(struct TreeNode* root, int val){
    if (root == NULL){
        return createNode(val);
    }
    if(val < root->val){
        root->left = insert(root->left, val);
    }
    else if(val > root->val){
        root->right = insert(root->right, val);
    }
    return root;
}
void inorderTraversal(struct TreeNode* root){
    if(root == NULL){
        return;
    }
    inorderTraversal(root->left);
    printf("%d", root->val);
    inorderTraversal(root->right);
}
void preorderTraversal(struct TreeNode* root){
    if(root == NULL){
        return;
    }
    printf("%d", root->val);
    preorderTraversal(root->left);
```

Date __/__/__
Page _____

```
    preorderTraversal(root->right);
}
void postorderTraversal(struct TreeNode* root){
    if(root==NULL){
        return;
    }
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d", root->val);
}
void displayTree(struct TreeNode* root){
    printf("Elements in tree:");
    inorderTraversal(root);
    printf("\n");
}
int main(){
    struct TreeNode* root=NULL;
    int choice=-1, val;
    printf("\n1. Insert element\n");
    printf("2. Display tree(inorder)\n");
    printf("3. Display tree(preorder)\n");
    printf("4. Display tree(postorder)\n");
    printf("5. Exit\n");
    while(choice!=5){
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter value to insert:");
                scanf("%d", &val);
                root=insert(root, val);
                break;
            case 2:
```

```
            display_tree(root);
            break;
        case 3:
            printf("elements in tree (preorder):");
            preorderTraversal(root);
            printf("\n");
            break;
        case 4:
            printf("elements in tree (postorder):");
            postorderTraversal(root);
            break;
        case 5:
            return 0;
    }
}
```

O/P

```
Enter your choice: 1
Enter value to insert: 33
Enter your choice: 1
Enter value to insert: 44
Enter your choice: 2
Elements in tree: 33 44
Enter your choice: 3
Elements in tree (preorder): 33 44
Enter your choice: 5
Exit.
```

LAB-7

Date __/__/__
Page _____

Q-1 Implement DFS traversal.

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 20
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES]; int n;
void dfs(int start){
    int i;
    visited[start] = 1;
    printf("visited %d \n", start);
    for(i=0; i<n; i++){
        if(graph[start][i] && !visited[i]){
            dfs[i];
        }
    }
}
int main(){
    int i, j, start;
    printf("Enter the number of vertices:");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter the starting vertex for DFS:");
    scanf("%d", &start);
    for(i=0; i<n; i++){
        visited[i]=0;
    }
    dfs(start);
```

```
    return 0;
}
```

O/P
```
Enter the number of vertices: 4
Enter the adjacency matrix:
1 0 0 1
0 1 0 0
1 1 1 0
1 0 0 0

Entering the starting vertex for DFS: 0
visited 0
visited 3
```

Q.2 Implement BFS.

I/P
```
#include<stdio.h>
void bfs(int a[10][10], int n, int v){
    int f, r, q[10], v;
    int s[10] = {0};
    printf("The nodes visited from %d", v);
    f = 0;
    r = -1;
    q[++r] = v;
    s[v] = 1;
    printf("%d", v);
    while (f <= r){
        v = q[f++];
        for (v=0; v<n; v++){
            if ((a[v][v]==1) && (s[v]==0)){
                printf("%d", v);
                s[v] = 1;
                q[++r] = v;}
}}
```

```
    printf("\n");
}
void main() {
    int n, a[10][10], source, i, j;
    printf("\n Enter no. of nodes:");
    scanf("%d", &n);
    printf("\n Enter the adjacency matrix:\n");
    for(i=0; i<n; i++) {
        for(j=0; j<n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    for(source=0; source<n; source++) {
        bfs(a, n, source);
    }
}
```

O/P

```
Enter no. of nodes: 3
Enter the adjacency matrix:
1 1 1
1 0 0
0 1 0
The node visited from 0   0 1 2
The node visited from 1   1 0 2
The node visited from 2   2 1 0.
```

Q-3 Delete a node in BST => Leetcode

I/P

```
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if(root == NULL)
        return root;
    if(key < root->val) {
        root->left = deleteNode(root->left, key);
```

```
else if (key > root->val)
    root->right = deleteNode(root->right, key);
else {
    if (root->left == NULL) {
        struct TreeNode* temp = root->right;
        free(root);
        return temp; }
    else if (root->right == NULL) {
        struct TreeNode* temp = root->left;
        free(root);
        return temp; }
    struct TreeNode* temp = minValueNode(root->right);
    root->val = temp->val;
    root->right = deleteNode(root->right, temp->val);
    }
    return root;
}
void inorder(struct TreeNode* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d", root->val);
        inorder(root->right);
    }
}
struct TreeNode* minValueNode(struct TreeNode* node) {
    struct TreeNode* current = node;
    while(current && current->left != NULL) {
        current = current->left; }
    return current;
}
```

Q4 Bottom left tree value -> leet code.

Date __/__/__
Page _____

```
int findBottomLeftValue(struct TreeNode* root) {
    struct TreeNode* queue[100];
    int front=0, rear=0;
    queue[rear++] = root;
    int leftmostValue = root->val;
    while (front < rear) {
        int levelSize = rear - front;
        for (int i=0; i<levelSize; i++) {
            struct TreeNode* current = queue[front++];
            if (i==0) {
                leftmostValue = current->val;
            }
            if (current->left != NULL) {
                queue[rear++] = current->left;
            }
            if (current->right != NULL) {
                queue[rear++] = current->right;
            }
        }
    }
    return leftmostValue;
}
```

Sem
4/3/24