PRACTICAL NO :- 3

Input -

```
/*Implement Circular Queue using Array. Perform following operations on it.
a) Insertion (Enqueue)
b) Deletion (Dequeue)
c) Display
*/
#include <iostream>
using namespace std;
int cqueue[5];
int front = -1 ,rear=-1,n=5;
void insert(int val){
if((front == 0 && rear ==n-1) || (front == rear+1)){
cout<<"\nQueue is filled "<<endl;
return;
}
if(front== -1){
front = 0;
rear = 0;
}
else{
if(rear==n-1){
rear=0;
}
else{
rear = rear+1;
}
}
cqueue[rear] = val;
}
void deletion(){
```

```
if(front == -1){
cout<<"\nQueue is already empty "<<endl;
return;
} cout<<"Element deleted from queue is --> "<<cqueue[front]<<endl;
if(front == rear){
front = -1;
rear = -1;
}
else{
if(front == n-1){
front = 0;
}
else{
front = front +1;
}
}
}
void display_front(){
int f= front ,r = rear;
if(front == -1){
cout<<"\nQueue is already empty "<<endl;
return ;
}
cout<<"Queue elements in forward order -->"<<endl;
if(f<=r){
while(f<=r){
cout<<cqueue[f]<<" ";
f++;
}
}
else{
```

```
while(f<=n-1){ cout<<cqueue[f]<<" ";
f++;
}
f=0;
while(f<=r){
cout<<cqueue[f]<<" ";
f++;
}
}
cout<<endl;
}
void display_reverse(){
int f= front ,r = rear;
if(front == -1){
cout<<"\nQueue is already empty "<<endl;
return ;
}
cout<<"Queue elements in reverse order -->"<<endl;
if(f<=r){
while(f<=r){
cout<<cqueue[r]<<" ";
r--;
}
}
else{
while(r>=0){
cout<<cqueue[r]<<" ";
r--;
} r=n-1;
while(r>=f){
cout<<cqueue[r]<<" ";
```

```
r--;
}
}
cout<<endl;
}
int main()
{
int ch,val;
cout<<"1]Insert"<<endl;
cout<<"2)Delete"<<endl;
cout<<"3)Display Forward"<<endl;
cout<<"4)Display Reverse"<<endl;
cout<<"5)Exit"<<endl;
do {
cout<<"Enter choice --> ";
cin>>ch;
switch(ch) {
case 1:
cout<<"Input for insertion--> ";
cin>>val;
cout<<endl;
insert(val);
break;
case 2:
deletion(); cout<<endl;
break;
case 3:
display_front();
cout<<endl;
break;
case 4:
```

```
display_reverse();

cout<<endl;

break;

case 5:

cout<<"Exit\n";

break;

default: cout<<"\nEnter correct choice !"<<endl;

}

} while(ch != 5);

return 0;

}
```

Output –

```
C:\Users\SANIKA>.\a.exe
1]Insert
2)Delete
3)Display Forward
4)Display Reverse
5)Exit
Enter choice --> 1
Input for insertion--> 10

Enter choice --> 1
Input for insertion--> 20

Enter choice --> 1
Input for insertion--> 30

Enter choice --> 3
Queue elements in forward order -->
10 20 30

Enter choice --> 4
Queue elements in reverse order -->
30 20 10

Enter choice --> 2
Element deleted from queue is --> 10

Enter choice --> 3
Queue elements in forward order -->
20 30

Enter choice --> 4
Queue elements in reverse order -->
30 20

Enter choice --> 5
Exit
```

PRACTICAL NO :- 4

Input –

//Construct an Expression Tree from prefix expression. Perform recursive and non- recursive In-order, pre□order and post-order traversals.

```
#include <iostream>

using namespace std;

typedef struct node //structure defined for node

{

char data;

struct node *left;

struct node *right;

} node;

typedef struct stacknode //structure defined for stack

{

node *data;

struct stacknode *next;

} stacknode;

class stack

{

stacknode *top; //top node is introduced

public:

stack()

{

top = NULL;

}

node *topp() //it will return the top element

{

return (top->data);

}
```

```
int isempty() //check if stack is empty(1)
{
if (top == NULL)
return 1;
return 0;
}
void push(node *a) //push function
{
stacknode *p;
p = new stacknode();
p->data = a;
p->next = top;
top = p;
}
node *pop() //pop function
{
stacknode *p;
node *x;
x = top->data;
p = top;
top = top->next;
return x;
}
};
node *create_pre(char prefix[10]);
node *create_post(char postfix[10]);
void inorder(node *p);
void preorder(node *p);
void postorder(node *p);
```

```
void inorder_non_recursive(node *t);
void preorder_non_recursive(node *t);
void postorder_non_recursive(node *t);
node *create_post(char postfix[10])
{
node *p;
stack s;
for (int i = 0; postfix[i] != '\0'; i++)
{
char token = postfix[i]; //token is the element in postfix55
if (isalnum(token)) //check if token is alphanumeric (operand)
{
p = new node(); //node creation
p->data = token;
p->left = NULL;
p->right = NULL;
s.push(p);
}
else //operator
{
p = new node();
p->data = token;
p->right = s.pop();
p->left = s.pop();
s.push(p);
}
}
return s.pop();
}
```

```
node *create_pre(char prefix[10])
{
node *p;
stack s;
int i;
for (i = 0; prefix[i] != '\0'; i++)
{
}
i = i - 1;
for (; i >= 0; i--)
{
char token = prefix[i]; // prefix element
if (isalnum(token)) // operand
{ //node creation
p = new node();
p->data = token;
p->left = NULL;
p->right = NULL;
s.push(p);
}
else //operator56
{
p = new node();
p->data = token;
p->left = s.pop();
p->right = s.pop();
s.push(p);
}
}
```

```
return s.pop();
}
void inorder(node *p) //inorder traversal using recursion
{
if (p == NULL)
{
return;
}
inorder(p->left);
cout << p->data;
inorder(p->right);
}
void preorder(node *p) //preorder traversal using recursion
{
if (p == NULL)
{
return;
}
cout << p->data;
preorder(p->left);
preorder(p->right);
}
void postorder(node *p) //postorder traversal using recursion
{
if (p == NULL)
{
return;
}
postorder(p->left);
```

```
postorder(p->right);
cout << p->data;
}
int main()
{
node *r = NULL, *r1;
char postfix[10], prefix[10];
int x;
int ch, choice;
do
{
cout << "\n\t*********MENU**********\n\n1.Construct tree from postfix Expression/prefix Expression.\n2.Inorder traversal.\n3.Preorder traversal.\n4.Postorder Traversal.\n5.Exit\n\nEnter your choice: ";
cin >> ch;
switch (ch)
{
case 1:
cout << "\nENTER CHOICE:\n\t1.Postfix expression\n\t2.Prefix expression\nChoice= ";
cin >> choice;
if (choice == 1)
{
cout << "\nEnter postfix expression= ";
cin >> postfix;
r = create_post(postfix);
}
else
{
cout << "\nEnter prefix expression= ";
cin >> prefix;
```

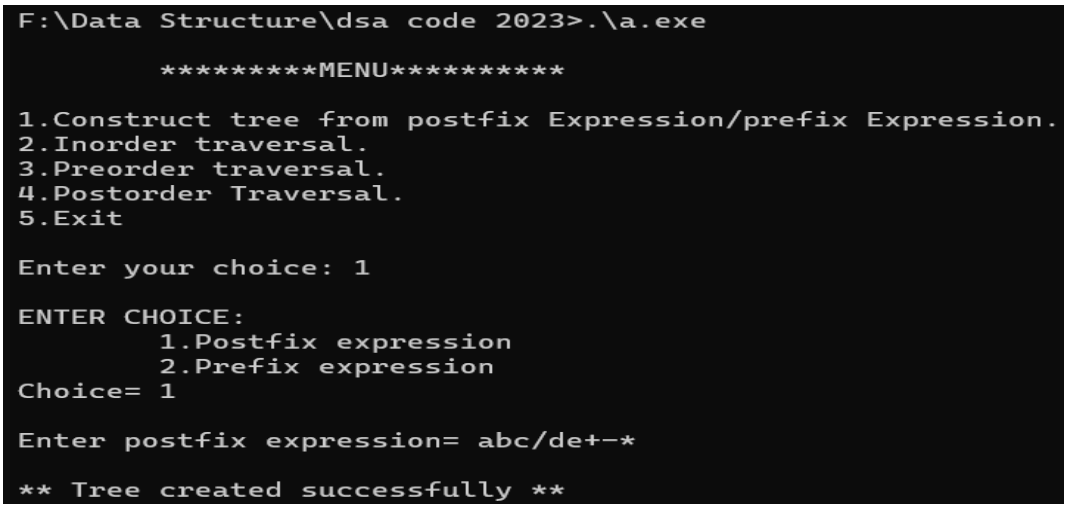
```
r = create_pre(prefix);
}
cout << "\n** Tree created successfully ** \n";
break;
case 2:
cout << "\n*********************" << endl;
cout << "\nInorder Traversal of tree\n\n";
cout << "With recursion:\t";
inorder(r);
cout << "\n\nWithout recursion: ";
inorder_non_recursive(r);
cout << "\n\n*********************" << endl;
break;
case 3:
cout << "*********************" << endl;
cout << "\nPreorder Traversal of tree\n\n";
cout << "With recursion:\t";
preorder(r);
cout << "\n\nWithout recursion: ";
preorder_non_recursive(r);
cout << "\n\n*********************" << endl;
break;
case 4:
cout << "*********************" << endl;
cout << "\nPostorder Traversal of tree\n\n";
cout << "With recursion:\t";
postorder(r);
cout << "\n\nWithout recursion: ";
postorder_non_recursive(r);
```

```
cout << "\n\n**********" << endl;
break;
}
} while (ch != 5);
return 0;
}
void inorder_non_recursive(node *t)
{
stack s;
while (t != NULL)
{ //data pushed in stack and moved to left till null(last)
s.push(t);
t = t->left;
}
while (s.isempty() != 1)
{
t = s.pop(); // topmost data of stack is printed and then moved to the right
cout << t->data;
t = t->right;
while (t != NULL)
{ //if child is represent push it to the stack
s.push(t);
t = t->left;
}
}
}
void preorder_non_recursive(node *t)
{
stack s; //stack
```

```
while (t != NULL)
{ //it will start from the root and then move to left
cout << t->data;
s.push(t);
t = t->left;
} //once left side is traversed we will pop and move to right
while (s.isempty() != 1)
{
t = s.pop();
t = t->right;
while (t != NULL)
{ //if child is represent we will push in stack
cout << t->data;
s.push(t);
t = t->left;
}
}
}
void postorder_non_recursive(node *t)
{
stack s, s1; //two stack maintained
node *t1; //root
while (t != NULL)
{
s.push(t);
s1.push(NULL);
t = t->left;
}
while (s.isempty() != 1)
```

```
{
t = s.pop();
t1 = s1.pop();
if (t1 == NULL)
{
s.push(t);
s1.push((node *)1);
t = t->right;
while (t != NULL)
{
s.push(t);
s1.push(NULL);
t = t->left;
}
}
else
cout << t->data;
}
}
```

Output–



```
        *********MENU**********

1.Construct tree from postfix Expression/prefix Expression.
2.Inorder traversal.
3.Preorder traversal.
4.Postorder Traversal.
5.Exit

Enter your choice: 2

*********************

Inorder Traversal of tree

With recursion: a*b/c-d+e

Without recursion: a*b/c-d+e

*********************

        *********MENU**********

1.Construct tree from postfix Expression/prefix Expression.
2.Inorder traversal.
3.Preorder traversal.
4.Postorder Traversal.
5.Exit

Enter your choice: 3
*********************

Preorder Traversal of tree

With recursion: *a-/bc+de

Without recursion: *a-/bc+de

*********************
```

```
        *********MENU**********

1.Construct tree from postfix Expression/prefix Expression.
2.Inorder traversal.
3.Preorder traversal.
4.Postorder Traversal.
5.Exit

Enter your choice: 4
**********************

Postorder Traversal of tree

With recursion: abc/de+-*

Without recursion: abc/de+-*

**********************

        *********MENU**********

1.Construct tree from postfix Expression/prefix Expression.
2.Inorder traversal.
3.Preorder traversal.
4.Postorder Traversal.
5.Exit

Enter your choice: 5

F:\Data Structure\dsa code 2023>
```

PRACTICAL NO :- 5

Input –

```
/*Implement binary search tree and perform following operations:
a) Insert (Handle insertion of duplicate entry)
b) Delete
c) Search
d) Display tree (Traversal)
e) Display - Depth of tree
f) Display - Mirror image
g) Create a copy
h) Display all parent nodes with their child nodes
i) Display leaf nodes
j) Display tree level wise
*/
#include <iostream>
#include<stdlib.h>
using namespace std;
struct node
{
int data;
struct node *left;
struct node *right;
};
node *insert(node *root, int val){
if (root == NULL){
node *temp; //new node temp
temp=new node;
temp->data=val;
temp->left=temp->right=NULL; //left and right is NULL bcz only one
```

```
node create;
return temp; // return single node
}
if (val < root->data){
root->left = insert(root->left, val);
}
else{
//val>root->data
root->right = insert(root->right, val);
}
return root;
}
void inorder(node *root)
{
if (root == NULL){
return;
}
inorder(root->left);
cout << root->data << " ";
inorder(root->right);
}
node* inorderSucc(node* root){
node* curr=root;
while(curr && curr->left!=NULL){
curr=curr->left;
}
return curr;
}
node *delet(node *root, int key){
```

```
if(key<root->data){
root->left=delet(root->left, key);
}
else if(key>root->data){
root->right=delet(root->right,key);
}
//if key==root->data
else{
if(root->left==NULL){
node* temp=root->right;
free(root);
return temp;
}
else if(root->right==NULL){
node* temp=root->left;
free(root);
return temp;
}
node* temp=inorderSucc(root->right);
root->data=temp->data;
root->right=delet(root->right, temp->data);
}
return root;
}
node *search(node* root, int val){
if(root==NULL){
return NULL;
}
if(val>root->data){
```

```
return search(root->right, val);
}
else if(val<root->data){
return search(root->left, val);
}
else{
return root;
}
}
void mirrorImg(node* root){
if(root==NULL){
return;
}
else{
struct node *temp;
mirrorImg(root->left);
mirrorImg(root->right);
swap(root->left,root->right);
}
}
node *copy(node *root){
node *temp=NULL;
if(root!=NULL){
temp=new node();
temp->data=root->data;
temp->left=copy(root->left);
temp->right=copy(root->right);
}
return temp;
```

```
}
void leafNodes(node* root){
if(root==NULL){
return;
}
if(!root->left && !root->right){
cout<<root->data<<" ";
return;
}
if(root->right)
leafNodes(root->right);
if(root->left)
leafNodes(root->left);
}
int calHeight(node* root){
if(root==NULL){
return 0;
}
int lheight=calHeight(root->left);
int rheight=calHeight(root->right);
return max(lheight,rheight)+1;
}
node *findMin(node *root){
if(root==NULL){
return NULL;
}
if(root->left)
return findMin(root->left);
else
```

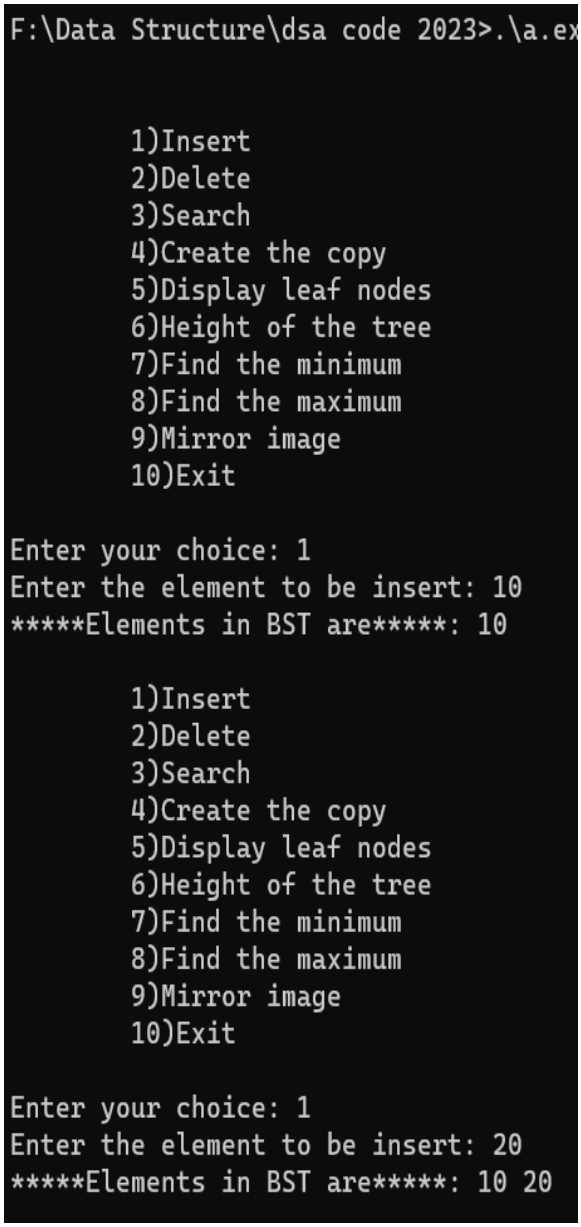
```
return root;
}
node *findMax(node *root){
if(root==NULL){
return NULL;
}
if(root->right)
return findMax(root->right);
else
return root;
}
int main()
{
node *root=NULL, *temp; //initially tree is NULL
int ch;
while (1){
cout<<"\n\n\t1)Insert" << endl;
cout<<"\t2)Delete" << endl;
cout<<"\t3)Search" << endl;
cout<<"\t4)Create the copy "<<endl;
cout<<"\t5)Display leaf nodes "<<endl;
cout<<"\t6)Height of the tree"<<endl;
cout<<"\t7)Find the minimum"<<endl;
cout<<"\t8)Find the maximum"<<endl;
cout<<"\t9)Mirror image"<<endl;
cout<<"\t10)Exit"<<endl;
cout<<"\nEnter your choice: ";
cin>>ch;
switch (ch){
```

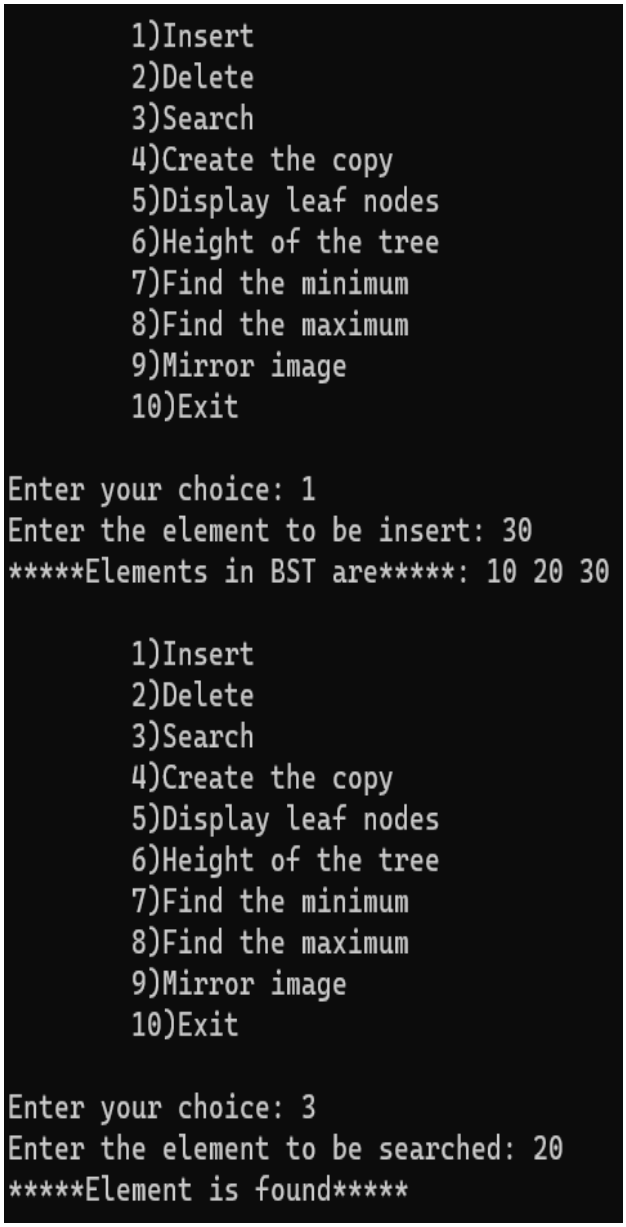
```
case 1:
cout << "Enter the element to be insert: ";
cin >> ch;
root= insert(root, ch);
cout << "*****Elements in BST are*****: ";
inorder(root);
break;
case 2:
cout<<"Enter the element to be deleted: ";
cin>>ch;
root=delet(root, ch);
cout<<"Element deleted successfully !!";
cout<<"\n*****After deletion the elements in the BST are*****: ";
inorder(root);
break;
case 3:
cout<<"Enter the element to be searched: ";
cin>>ch;
temp=search(root, ch);
if(temp==NULL){
cout<<"*****Element is not found*****";
}
else{
cout<<"*****Element is found*****";
}
break;
case 4:
cout<<"The copy of the tree is: ";
root=copy(root);
```

```
inorder(root);
break;
case 5:
cout<<"The leaf nodes are: ";
leafNodes(root);
break;
case 6:
cout<<"Height of the binary search tree is: "<<calHeight(root);
break;
case 7:
temp=findMin(root);
cout<<"\nMinimum element is : "<<temp->data;
break;
case 8:
temp=findMax(root);
cout<<"\nMaximum element is: "<<temp->data;
break;
case 9:
cout<<" inorder tree: ";
inorder(root);
cout<<endl;
mirrorImg(root);
cout<<"mirror image is: ";
inorder(root);
break;
case 10:
return 0;
default:
cout<<"\nInvalid choice !! Please enter your choice again";
```

```
}

}

return 0;

}
```

Output –





```
        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 2
Enter the element to be deleted: 10
Element deleted successfully !!
*****After deletion the elements in the BST are*****: 20 30

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 4
The copy of the tree is: 20 30

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit
```

```
        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 5
The leaf nodes are: 30

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 6
Height of the binary search tree is: 2

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit
```

```
Enter your choice: 1
Enter the element to be insert: 70
*****Elements in BST are*****: 20 30 70

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 7

Minimum element is : 20

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 8
```

```
Maximum element is: 70

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 9
 inorder tree: 20 30 70
mirror image is: 70 30 20

        1)Insert
        2)Delete
        3)Search
        4)Create the copy
        5)Display leaf nodes
        6)Height of the tree
        7)Find the minimum
        8)Find the maximum
        9)Mirror image
        10)Exit

Enter your choice: 10

F:\Data Structure\dsa code 2023>
```

PRACTICAL NO :- 6

Input –

```
//Implement In-order Threaded Binary Tree and traverse it in In-order and Pre-order.
#include<bits/stdc++.h>
using namespace std;
class Node{
public:
int data;
Node* left;
Node* right;
int leftThread; // leftThread=0 -> left pointer points to the inorder predecessor
int rightThread; // rightThread=0 -> right pointer points to the inorder successor
Node(int val){
this->data = val;
}
};
class DoubleThreadedBinaryTree{
private:
Node* root;
public:
DoubleThreadedBinaryTree(){
// dummy Node with value as INT_MAX
root = new Node(INT_MAX);
root->left = root->right = root;
root->leftThread = 0;
root->rightThread = 1;
}
void insert(int data){
Node* new_node = new Node(data);
```

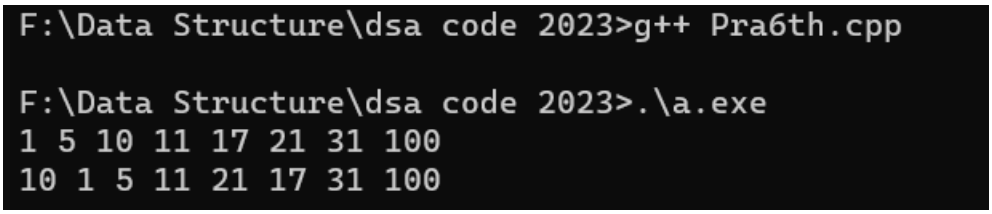
```
if(root->left == root && root->right == root){
//Empty Tree
new_node->left = root;
root->left = new_node;
new_node->leftThread = 0;
new_node->rightThread = 0;
root->leftThread = 1;
new_node->right = root;
return;
}
else{
Node* current = root->left;
while(true){
if(current->data > data){
if(current->leftThread == 0 ){
// this is the last Node
new_node->left = current->left;
current->left = new_node;
new_node->leftThread = current->leftThread;
new_node->rightThread = 0;
current->leftThread = 1;
new_node->right = current;
break;
}
else{
current = current->left;
}
}
else{
```

```
if(current->rightThread == 0){
// this is the last Node
new_node->right = current->right;
current->right = new_node;
new_node->rightThread = current->rightThread;
new_node->leftThread = 0;
current->rightThread=1;
new_node->left = current;
break;
}
else{
current = current->right;
}
}
}
}
}
Node* findNextInorder(Node* current){
if(current->rightThread == 0){
return current->right;
}
current = current->right;
while (current->leftThread != 0)
{
current = current->left;
}
return current;
}
void inorder(){
```

```
Node* current = root->left;
while(current->leftThread == 1){
current = current->left;
}
while(current != root){
cout<<current->data<<" ";
current = findNextInorder(current);
}
cout<<"\n";
}
void preorder(){
Node* current = root->left;
while(current != root){
cout<<current->data<<" ";
if(current->left != root && current->leftThread != 0)
current= current->left;
else if(current->rightThread == 1){
current = current->right;
}
else{
while (current->right != root && current->rightThread == 0)
{
current = current->right;
}
if(current->right == root)
break;
else
{
current=current->right;
```

```
}
}
}
cout<<"\n";
}
};
int main(){
DoubleThreadedBinaryTree dtbt;
dtbt.insert(10);
dtbt.insert(1);
dtbt.insert(11);
dtbt.insert(5);
dtbt.insert(21);
dtbt.insert(17);
dtbt.insert(31);
dtbt.insert(100);
dtbt.inorder();
dtbt.preorder();
return 0;
}
```

Output-



PRACTICAL NO :- 7

Input –

/*Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find minimum spanning tree a) Using Prim's algorithm.primes

*/

```
#include <iostream>

using namespace std;

class graph

{

int G[20][20], n;

public:

void accept()

{

int i, j, e;

int src, dest, cost;

cout << "\nEnter the no. of vertices: ";

cin >> n;

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

for (j = 0; j < n; j++){

G[i][j] = 0;

}

}

cout << "\nEnter the no. of Edges: ";

cin >> e;

for (i = 0; i < e; i++){

cout << "\nEnter Source: ";

cin >> src;

cout << "\nDestination: ";

cin >> dest;
```

```
cout << "\nCost: ";
cin >> cost;
G[src][dest] = cost;
G[dest][src] = cost;
}
}
void display(){
int i, j;
for (i = 0; i < n; i++){
cout << "\n";
for (j = 0; j < n; j++){
cout << "\t" << G[i][j];
}
}
}
void prims()
{
int i, j, R[20][20];
int src, dest, cost, count, min;
int total = 0;
int visited[20];
for (i = 0; i < n; i++){
for (j = 0; j < n; j++){
if (G[i][j] == 0){
R[i][j] = 999;
}
else
R[i][j] = G[i][j];
}
```

```
}
for (i = 0; i < n; i++) {
visited[i] = 0;
}
cout << "\nEnter start vertex: ";
cin >> src;
visited[src] = 1;
count = 0;
while (count < n - 1) {
min = 999;
for (i = 0; i < n; i++){
if (visited[i] == 1)
for (j = 0; j < n; j++){
if (visited[j] != 1){
if (min > R[i][j]){
min = R[i][j];
src = i;
dest = j;
}
}
}
}
cout << "\nEdge from " << src << " to " << dest << " \twith cost: " <<
min;
total = total + min;
visited[dest] = 1;
count++;
}
cout << "\nTotal Cost: " << total << "\n";
```

```
}
};
int main()
{
graph g;
g.accept();
g.display();
g.prims();
}
```

Output –

```
F:\Data Structure\dsa code 2023>g++ Pra7th.cpp

F:\Data Structure\dsa code 2023>.\a.exe

Enter the no. of vertices: 6

Enter the no. of Edges: 8

Enter Source: 1

Destination: 5

Cost: 20

Enter Source: 3

Destination: 6

Cost: 30

Enter Source: 2

Destination: 4

Cost: 10
```

```
Enter Source: 5

Destination: 6

Cost: 5

Enter Source: 4

Destination: 6

Cost: 20

Enter Source: 3

Destination: 5

Cost: 10

Enter Source: 1

Destination: 6

Cost: 40

Enter Source: 2

Destination: 3

Cost: 32
```

```
        0       0       0       0       0       0
        0       0       0       0       0       20
        0       0       0       32      10      0
        0       0       32      0       0       10
        0       0       10      0       0       0
        0       20      0       10      0       0
Enter start vertex: 2

Edge from 2 to 4        with cost: 10
Edge from 2 to 3        with cost: 32
Edge from 3 to 5        with cost: 10
Edge from 5 to 1        with cost: 20
Edge from 5 to 1        with cost: 999
Total Cost: 1071

F:\Data Structure\dsa code 2023>
```

PRACTICAL NO :- 8

Input –

/*Represnt a graph of your college campus using adjacency list/adjacency matrix.Nodes should represnt a various departments and link shold represntthe distance between them.Find minimum spanning tree using Krusakls alogrithm

*/

```
#include <iostream>

using namespace

std; class graph{

int g[20][20];

int e,v;

public:

 void accept(); void

display(); void

dijkstra(int start);

}; void graph:: accept(){ int src, dest,

cost, i,j; cout<<"Enter the number of vertices: "; cin>>v;

 cout<<"Enter the number of edges: ";

cin>>e; for(i=0; i<v; i++){

for(j=0; j<v;j++){

g[i][j]=0;

 } } for(i=0; i<e; i++){

cout<<"\nEnter source vertex: ";

cin>>src;

 cout<<"Enter destination vertex: ";

cin>>dest;

 cout<<"Enter the cost of the edge: ";

cin>>cost; g[src][dest]=cost;

g[dest][src]=cost;

 } } void
```

```
graph::display(){
int i,j; for(i=0;
i<v; i++){
cout<<endl;
for(j=0; j<v; j++){
cout<<g[i][j]<<"\t";
 }
 } }
void graph::dijkstra(int start){
int r[20][20],
visited[20],distance[20],from[20],i,j,cnt,mindst,next;
for(i=0; i<v; i++){ for(j=0; j<v; j++){
if(g[i][j]==0){ r[i][j]=999;
 }
else{
 r[i][j]=g[i][j];
}
 } } for(i=0;
i<v; i++){ visited[i]=0;
from[i]=start;
distance[i]=r[start][i];
 }
 distance[start]=0;
visited[start]=1;
cnt=v; while(cnt>0){
mindst=999;
for(i=0; i<v; i++){
 if((mindst>distance[i]) && visited[i]==0){
mindst=distance[i]; next=i;
```

```
 } }
visited[next]=1; for(i=0; i<v; i++){
if(visited[i]==0 &&
distance[i]>(mindst+r[next][i])){
distance[i]=mindst+r[next][i];
from[i]=next;
 }
} cnt--
; }
 for(i=0; i<v; i++){
 cout<<"\nDistance of "<<i<< " from "<<start<<" is "<<distance[i]<<endl<<"Path "<<i;
j=i; do{
j=from[j]; cout<<"<- "<<j;
 }
 while(j!=start);
 } } int
main() {
graph g;
int s;
 g.accept();
 g.display();
 cout<<"\nEnter the starting vertex: ";
cin>>s;
 g.dijkstra(s); return
0;
}
```

Output –

```
F:\Data Structure\dsa code 2023>g++ Pra8th.cpp

F:\Data Structure\dsa code 2023>.\a.exe
Enter the number of vertices: 4
Enter the number of edges: 6

Enter source vertex: 1
Enter destination vertex: 4
Enter the cost of the edge: 10

Enter source vertex: 4
Enter destination vertex: 4
Enter the cost of the edge: 5

Enter source vertex: 2
Enter destination vertex: 4
Enter the cost of the edge: 20

Enter source vertex: 3
Enter destination vertex: 4
Enter the cost of the edge: 9

Enter source vertex: 2
Enter destination vertex: 2
Enter the cost of the edge: 15
```

```
Enter source vertex: 1
Enter destination vertex: 3
Enter the cost of the edge: 20

0       0       0       0
0       0       0       20
0       0       15      0
0       20      0       0
Enter the starting vertex: 3

Distance of 0 from 3 is 999
Path 0<- 3
Distance of 1 from 3 is 20
Path 1<- 3
Distance of 2 from 3 is 999
Path 2<- 3
Distance of 3 from 3 is 0
Path 3<- 3
F:\Data Structure\dsa code 2023>
```

PRACTICAL NO :- 9

Input –

```
//Implement Heap sort to sort given set of values using max or min heap.
#include <iostream>
using namespace std;
void maxHeapify(int a[], int i, int n){
int j, temp;
temp=a[i];
j=2*i;
while(j<=n){
if(j<n && a[j+1]>a[j])
j=j+1;
if(temp>a[j])
break;
else if(temp<=a[j]){
a[j/2]=a[j];
j=2*j;
}
}
a[j/2]=temp;
return;
}
void build_maxheap(int a[], int n){
int i;
for(i=n/2 ; i>=1; i--){
maxHeapify(a,i,n);
}
}
void max_HeapSort(int a[], int n){
```

```
int i, temp;
for(i=n; i>=2; i--){
temp = a[i];
a[i] = a[1];
a[1] = temp;
maxHeapify(a, 1, i-1);
}
}
void min_heapify(int a[], int i, int n){
int j, temp;
temp = a[i];
j = 2*i;
while(j<=n){
if(j<n && a[j+1]<a[j])
j=j+1;
if(temp<a[j])
break;
else if(temp>=a[j]){
a[j/2] = a[j];
j= 2*j;
}
}
a[j/2] = temp;
return;
}
void build_minheap(int a[], int n){
int i;
for(i=n/2; i>=1; i--){
min_heapify(a,i,n);
```

```
}
}
void min_HeapSort( int a[], int n){
int i, temp;
for(i=n; i>=2; i--){
temp = a[i];
a[i] = a[1];
a[1] = temp;
min_heapify(a, 1, i-1);
}
}
void print(int arr[], int n){
cout<<"\nsorted data: ";
for(int i=1; i<=n; i++){
cout<<"->"<<arr[i];
}
return;
}
int main()
{
int n, i, ch;
cout<<"Enter the number of elements to be sorted: " ;
cin>>n;
int arr[n];
for(i=1; i<=n; i++) {
cout<<"Enter element "<<i<<": ";
cin>>arr[i];
}
do{
```

```
cout<<"\n\n1]Heap sort using max heap";
cout<<"\n2]Heap sort using min heap";
cout<<"\n3]Exit";
cout<<"\nEnter your choice: ";
cin>>ch;
switch(ch){
case 1:
build_maxheap(arr, n);
max_HeapSort(arr, n);
print(arr, n);
break;
case 2:
build_minheap(arr, n);
min_HeapSort(arr, n);
print(arr, n);
break;
}
}while(ch!=3);
return 0;
}
```

Input –