1.)

#include<iostream>

using namespace std;

struct node{

    string data;

    node \*left;

    node \*right;

};

int index=0, ltree, rtree;

string treeLeaves[100];

node \*root=NULL;

node \*parents[100];

node \*parents1[100];

void createTree(int n){

    index=0;

    for(int i=1;i<n-1;i++){

        if(treeLeaves[i]!="(" && treeLeaves[i]!="," && treeLeaves[i]!=")"){

            node \*temp = new node;

            temp->data = treeLeaves[i];

            temp->right=NULL;

            temp->left=NULL;

            //case 1 children of left child

            if(treeLeaves[i-1]=="(" && treeLeaves[i+1]=="(" && root!=NULL){

                parents[index-1]->left=temp;

                parents[index]=temp;

                index++;

            }

            //case 2 children of right child

            if(treeLeaves[i-1]=="," && treeLeaves[i+1]=="(" && root!=NULL){

                parents[index-1]->right=temp;

                parents[index]=temp;

                index++;

            }

            //case 3 right child

            if(treeLeaves[i-1]=="," && treeLeaves[i+1]==")" && root!=NULL){

                parents[index-1]->right=temp;

                index--;

            }

            //case 4 left child

            if(treeLeaves[i-1]=="(" && treeLeaves[i+1]=="," && root!=NULL){

                parents[index-1]->left=temp;

            }

            //case 5 only child

            if(treeLeaves[i-1]=="(" && treeLeaves[i+1]==")" && root!=NULL){

                parents[index-1]->left=temp;

                index--;

            }

            if(root==NULL){

                root=temp;

                parents[index]=temp;

                index++;

            }

        }

        if(treeLeaves[i]==")" && treeLeaves[i+1]==")")

            index--;

    }

}

void traverseInOrderRecursion(node \*t){

    if(t==NULL)

        return;

    traverseInOrderRecursion(t->left);

    cout<<"->"<<t->data;

    traverseInOrderRecursion(t->right);

}

void traversePreOrderRecursion(node \*t){

    if(t==NULL)

        return;

    cout<<"->"<<t->data;

    traversePreOrderRecursion(t->left);

    traversePreOrderRecursion(t->right);

}

void traversePostOrderRecursion(node \*t){

    if(t==NULL)

        return;

    traversePostOrderRecursion(t->left);

    traversePostOrderRecursion(t->right);

    cout<<"->"<<t->data;

}

void traverseInOrderIterative(){

    index=-1;

    node \*temp = root;

    while(temp!=NULL || index!=-1){

        if(temp!=NULL){

            index++;

            parents1[index]=temp;

            temp=temp->left;

        }

        else{

            temp=parents1[index];

            index--;

            cout<<"->"<<temp->data;

            temp=temp->right;

        }

    }

}

void traversePreOrderIterative(){

    index=-1;

    node \*temp = root;

    index++;

    parents1[index]=temp;

    while (index>-1){

        temp = parents1[index];

        cout<<"->"<<temp->data;

        index--;

        if (temp->right){

            index++;

            parents1[index]=temp->right;

        }

        if (temp->left){

            index++;

            parents1[index]=temp->left;

        }

    }

}

void traversePostOrderIterative(){

    index=-1;

    index++;

    int indexPost = -1;

    node \*temp = root;

    parents1[index]=temp;

    string post[100];

    while(index>-1){

        temp = parents1[index];

        index--;

        indexPost++;

        post[indexPost]=temp->data;

        if(temp->left){

            index++;

            parents1[index]=temp->left;

        }

        if(temp->right){

            index++;

            parents1[index]=temp->right;

        }

    }

    while (indexPost>-1){

        cout<<"->"<<post[indexPost];

        indexPost--;

    }

}

int main(){

    int n;

    cout<<endl<<"Enter tree string size with parenthesis and comma : ";

    cin>>n;

    cout<<endl<<"Enter tree string one by one with parenthesis and comma";

    cout<<endl<<"If only right child of any present then enter as following";

    cout<<endl<<"Eg. if 6 has only right child 7 then enter 6(,7) :";

    cout<<endl;

    for(int i=0;i<n;i++)

        cin>>treeLeaves[i];

    createTree(n);

    cout<<endl<<"Inorder Recursion ";

    traverseInOrderRecursion(root);

    cout<<endl<<endl<<"Preorder Recursion ";

    traversePreOrderRecursion(root);

    cout<<endl<<endl<<"Postorder Recursion ";

    traversePostOrderRecursion(root);

    cout<<endl<<endl<<"Inorder Iterative ";

    traverseInOrderIterative();

    cout<<endl<<endl<<"Preorder Iterative ";

    traversePreOrderIterative();

    cout<<endl<<endl<<"Postorder Iterative ";

    traversePostOrderIterative();

}

2.)

//construct array from complete binary tree

#include<iostream>

using namespace std;

struct node{

    string data;

    node \*right;

    node \*left;

};

node \*root=NULL;             //root node of the tree since we have to create the tree not given then compile it in the array

node \*parents[100];          //a structure needed to create the tree

node \*queue[100];            //used to create array from created tree

string arrayOfTree[100];     //array of trees created

string treeLeaves[100];      //user input for the string of tree to costruct the tree and then to convertthe tree into array

int index=0;                 //for the array of tree

int front=-1, rear=-1;       //for the queue

int trace=0;

//left child of the element

void leftChild(string data){

    trace=0;

    for(int i=0;i<index;++i)

        if(arrayOfTree[i]==data){

            trace=i+1;

            break;

        }

    if(2\*trace<=index)

        cout<<endl<<"Left child of "<<data<<" is "<<arrayOfTree[2\*trace-1];

    else

        cout<<endl<<"No left child";

}

//right child of the element

void rightChild(string data){

    trace=0;

    for(int i=0;i<index;++i)

        if(arrayOfTree[i]==data){

            trace=i+1;

            break;

        }

    if(2\*trace+1<=index)

        cout<<endl<<"Right child of "<<data<<" is "<<arrayOfTree[(2\*trace)];

    else

        cout<<endl<<"No right child";

}

//parent of the element

void parent(string data){

    trace=0;

    for(int i=0;i<index;++i)

        if(arrayOfTree[i]==data){

            trace=i+1;

            break;

        }

    if(trace==1)

        cout<<endl<<"Root node no parent.";

    else

        cout<<endl<<"Parent of "<<data<<" is "<<arrayOfTree[trace/2-1];

}

//sibling of the element

void sibling(string data){

    trace=0;

    for(int i=1;i<index;++i)

        if(arrayOfTree[i]==data){

            trace=i;

            break;

        }

    if(trace>0 && trace<index){

        if(trace%2==0)

            cout<<endl<<"Sibling of "<<data<<" is "<<arrayOfTree[trace-1];

        else

            cout<<endl<<"Sibling of "<<data<<" is "<<arrayOfTree[trace+1];

    }

    else

        cout<<endl<<"Either no sibling or root node ";

}

//push data into the queue

void push(node \*item){

    if(rear == 99)

        cout<<endl<<" Queue is full ! ";

    else{

        rear++;

        queue[rear] = item;

        if(front == -1)

            front=0;

    }

}

//pop elements from the queue

void pop(){

    if(front == -1)

        cout<<endl<<"Queue is empty ! ";

    else{

        front++;

        if(front > rear){

            front = -1;

            rear = -1;

        }

    }

}

//check if the queue is empty

int isEmpty(){

    if(front==-1 || front>rear)

        return 1;

    else

        return 0;

}

//create the array of tree

int createArray(node \*t){

    index=0;

    node \*temp=t;

    push(temp);

    while(isEmpty()!=1){

        node \*tmp = queue[front];

        arrayOfTree[index] = tmp->data;

        index++;

        pop();

        if(tmp->left!=NULL)

            push(tmp->left);

        if(tmp->right!=NULL)

            push(tmp->right);

    }

}

//create tree of input string

void createTree(int n){

    index=0;

    for(int i=0;i<n;i++){

        if(treeLeaves[i]!="(" && treeLeaves[i]!="," && treeLeaves[i]!=")"){

            node \*temp = new node;

            temp->data = treeLeaves[i];

            temp->right=NULL;

            temp->left=NULL;

            //case 1 children of left child

            if(treeLeaves[i-1]=="(" && treeLeaves[i+1]=="(" && root!=NULL){

                parents[index-1]->left=temp;

                parents[index]=temp;

                index++;

            }

            //case 2 children of right child

            if(treeLeaves[i-1]=="," && treeLeaves[i+1]=="(" && root!=NULL){

                parents[index-1]->right=temp;

                parents[index]=temp;

                index++;

            }

            //case 3 right child

            if(treeLeaves[i-1]=="," && treeLeaves[i+1]==")" && root!=NULL){

                parents[index-1]->right=temp;

                index--;

            }

            //case 4 left child

            if(treeLeaves[i-1]=="(" && treeLeaves[i+1]=="," && root!=NULL){

                parents[index-1]->left=temp;

            }

            //case 5 only child

            if(treeLeaves[i-1]=="(" && treeLeaves[i+1]==")" && root!=NULL){

                parents[index-1]->left=temp;

                index--;

            }

            if(root==NULL){

                root=temp;

                parents[index]=temp;

                index++;

            }

        }

        if(treeLeaves[i]==")" && treeLeaves[i+1]==")")

            index--;

    }

}

void showArray(){

    cout<<endl<<"Showing array : ";

    for(int i=0;i<index;++i)

        cout<<arrayOfTree[i]<<"  ";

}

//main method

int main(){

    int n;

    cout<<endl<<"Enter tree string size with parenthesis and comma : ";

    cin>>n;

    cout<<endl<<"Enter sring of tree : "<<endl;

    for(int i=0;i<n;i++)

        cin>>treeLeaves[i];

    createTree(n);

    createArray(root);

    showArray();

    sibling("b");

    sibling("a");

    sibling("l");

    sibling("d");

    parent("a");

    parent("g");

    parent("j");

    rightChild("d");

    rightChild("a");

    rightChild("f");

    leftChild("e");

    leftChild("a");

    leftChild("f");

    return 0;

}

3.)

#include<iostream>

using namespace std;

struct node{

    int data;

    node \*left;

    node \*right;

};

node \*root = NULL;

int c=0;

void traverseInOrderRecursion(node \*t){

    if(t==NULL)

        return;

    traverseInOrderRecursion(t->left);

    cout<<"->"<<t->data;

    traverseInOrderRecursion(t->right);

}

void addLeaf(int dataI){

    node \*temp = new node;

    temp->data = dataI;

    temp->left=NULL;

    temp->right=NULL;

    if(root==NULL)

        root=temp;

    else{

        node \*tmp = root;

        node \*tempPrev;

        while(tmp!=NULL){

            if(temp->data<tmp->data){

                tempPrev=tmp;

                tmp=tmp->left;

                c=0;

            }

            else{

                tempPrev=tmp;

                tmp=tmp->right;

                c=1;

            }

        }

        if(c==0)

            tempPrev->left=temp;

        if(c==1)

            tempPrev->right=temp;

    }

}

int search(int data){

    node \*temp = root;

    node \*tempPrev;

    if(temp==NULL){

        cout<<endl<<"Empty tree";

        return -1;

    }

    else{

        while(temp!=NULL){

            if(data<temp->data){

                tempPrev=temp;

                temp=temp->left;

                if(temp==NULL)

                    break;

                c=0;

            }

            if(data>temp->data){

                tempPrev=temp;

                temp=temp->right;

                if(temp==NULL)

                    break;

                c=1;

            }

            if(data==temp->data){

                if(c==0){

                    cout<<"Found";

                    cout<<endl<<"Left child of "<<tempPrev->data;

                    return 1;

                }

                if(c==1){

                    cout<<"Found";

                    cout<<endl<<"Right child of "<<tempPrev->data;

                    return 1;

                }

            }

        }

        cout<<endl<<"Not found!";

        return -1;

    }

}

int main(){

    char flag='y';

    int input;

    cout<<endl<<"Enter data in BST : ";

    while(flag=='y'){

        cout<<endl<<"Enter data : ";

        cin>>input;

        addLeaf(input);

        cout<<endl<<"More data ? :";

        cin>>flag;

    }

    flag='y';

    cout<<endl<<"Inorder tree : just for ref :";

    traverseInOrderRecursion(root);

    while(flag=='y'){

        cout<<endl<<"Enter element to search : ";

        cin>>input;

        search(input);

        cout<<endl<<"More search ? ";

        cin>>flag;

    }

    return 0;

}