```
/*
 * BST.h
 *  Created on: Nov 8, 2020
 *      Author: hp
 */
#ifndef BST_H_
#define BST_H_
struct Node{
	int data;
	Node*left,*right;
};
class BST {
	Node*root;
public:
	BST();
	Node* create(int);
	bool insert(int);
	Node* getRoot();
	Node* deleteNode(Node*,int);
	void inorder(Node*);
	void preorder(Node*);
	void postorder(Node*);
	bool search(Node*,int);
	Node* findMin(Node*);
	int treeDepth(Node*);
	void mirrorImg(Node*);
	Node* createCopy(Node*);
```

```
        void printLeafNode(Node*);
        void printParentChild(Node*);
        void printLevelWise();
        virtual ~BST();
};
#endif /* BST_H_ */
```

- **BST.cpp**

```
/*
 * BST.cpp
 *  Created on: Nov 8, 2020
 *      Author: hp
 */

#include<iostream>
#include<cstdlib>
#include<queue>
#include "BST.h"
using namespace std;

BST::BST() {
        // TODO Auto-generated constructor stub
        root=NULL;
```

```
}

//-------------------------definition of creation of node-----------------------------
Node* BST::create(int num){
	Node* temp=(Node*)malloc(sizeof(Node));
	temp->data=num;
	temp->left=NULL;
	temp->right=NULL;
	return temp;
}
//-------------------------definition of method to insert new node-----------------------
bool BST::insert(int num){
	Node* newNode=create(num);
	if(root==NULL)
		root=newNode;
	else{
		Node*temp=root,*parent;
		while(temp!=NULL){
			parent=temp;
			if(temp->data == newNode->data)
				return false;
			if(newNode->data < temp->data)
				temp=temp->left;
			else
				temp=temp->right;
		}
		if(newNode->data < parent->data)
```

```
                parent->left=newNode;
            else
                parent->right=newNode;
        }
        return true;
}
//--------------------------------definition of getRoot method-------------------------------
Node* BST::getRoot(){
        return root;
}
//--------------------------------definition of inorder traversal-----------------------------
void BST::inorder(Node* temp){
        if(temp==NULL)
            return;
        inorder(temp->left);
        cout<<temp->data<<" || ";
        inorder(temp->right);
}
//--------------------------------definition of preorder traversal----------------------------
void BST::preorder(Node*temp){
        if(temp==NULL)
            return;
        cout<<temp->data<<" || ";
        preorder(temp->left);
        preorder(temp->right);
```

```
}
//---------------------------------definition of postorder travesal-----------------------------
---------
void BST::postorder(Node*temp){
	if(temp==NULL)
		return;
	postorder(temp->left);
	postorder(temp->right);
	cout<<temp->data<<" || ";
}
//---------------------------------definition of search ----------------------------------------
--------
bool BST::search(Node*temp,int num){
	while(temp!=NULL)
	{
		if(num < temp->data)
			temp=temp->left;
		else if(num > temp->data)
			temp=temp->right;
		else
			return true;
	}
	return false;
}
//-----------------------------definition of delete-----------------------------------------
------------
Node* BST::deleteNode(Node*T,int num){
```

```
if(T==NULL)
        return T;
//if num is smaller than temp node
if(num < T->data)
        T->left=deleteNode(T->left,num);
//if num is greater than temp node
else if(num > T->data)
        T->right=deleteNode(T->right,num);
//
else{
        Node*temp=T;
        //if node to be deleted is leaf node or node with one child
        if(T->left==NULL){
                T=T->right;
                free(temp);
                return T;
        }
        else if(T->right==NULL){
                T=T->left;
                free(temp);
                return T;
        }
        //if node has 2 children
        temp=findMin(T->right);
        T->data=temp->data;
        T->right=deleteNode(T->right,temp->data);
```

```
        }

        return T;
}
//-----------------------definition of function to find min value-----------------------------
Node* BST::findMin(Node*temp){
        while(temp->left!=NULL)
                temp=temp->left;
        return temp;
}
//---------------------------definition of depth method-------------------------------------
int BST::treeDepth(Node*temp){
        if(temp==NULL)
                return 0;
        return 1+ max(treeDepth(temp->left),treeDepth(temp->right));
}
//----------------------definition of mirror image method-----------------------------------
void BST::mirrorImg(Node*temp){
        if(temp==NULL)
                return;
        Node*T=temp->left;
        temp->left=temp->right;
        temp->right=T;
        mirrorImg(temp->left);
```

```
	mirrorImg(temp->right);
}
//----------------------definition of crete copy method-----------------------------------
--------------------
Node* BST::createCopy(Node* temp){
	if(temp==NULL)
		return NULL;
	Node* newNode=(Node*)malloc(sizeof(Node));
	newNode->left=createCopy(newNode->left);
	newNode->right=createCopy(newNode->right);
	return newNode;
}
//----------------------definition to display leaf nodes-----------------------------------
--------------------
void BST::printLeafNode(Node*temp){
	if(temp==NULL)
		return;
	if(temp->left==NULL && temp->right==NULL)
		cout<<temp->data<<" ";
	if(temp->left!=NULL)
		printLeafNode(temp->left);
	if(temp->right!=NULL)
		printLeafNode(temp->right);
}
//----------------------definition of display parent child---------------------------------
--------------------
void BST::printParentChild(Node*temp){
	if(temp==NULL)
```

```
            return;
        if(temp->left || temp->right)
        {
            cout<<"\tParent:"<<temp->data;
            cout<<"Child:";
            if(temp->left)
                cout<<temp->left->data<<" ";
            if(temp->right)
                cout<<temp->right->data<<" "<<endl;
            printParentChild(temp->left);
            printParentChild(temp->right);
        }
}
//-------------------------definition of level wise display-------------------------------
-------------------------
void BST::printLevelWise(){
        queue<Node*>q;
        q.push(root);
        q.push(NULL);
        while(q.size()>1){
            Node* current=q.front();
            q.pop();
            if(current==NULL){
                q.push(NULL);
                cout<<"\n";
            }
            else{
```

```
                if(current->left)
                        q.push(current->left);
                if(current->right)
                        q.push(current->right);
                cout<<current->data<<" ";
        }
    }

}
BST::~BST() {
    // TODO Auto-generated destructor stub
}
```

- **Assignment5.cpp**

```
//========================================================================
// Name        : Assignment5.cpp
// Author      : Megha Sonavane
// Description : Binary search Tree
//========================================================================

#include <iostream>
#include "BST.h"
#include "Queue.h"
//#include "BST.cpp"
using namespace std;
```

```
int main() {

	BST bst;
	Node* temp,*cpy;
	int ch,num;
	bool flag;

	cout<<"\tBinary Search Tree";
	do{

	cout<<endl<<"================================================
================================================================
=============="<<endl;
		cout<<"\t1:Insert node into tree"<<endl<<"\t2:Delete
"<<endl<<"\t3:Searching"<<endl<<"\t4:Traversal"<<endl<<"\t5:Depth of
tree"<<endl;
		cout<<"\t6:Create mirror image"<<endl<<"\t7:Create
copy"<<endl<<"\t8:Display all perent nodes with child"<<endl<<"\t9:Display leaf
nodes"<<endl<<"\t10:Display tree level wise"<<endl<<"\t0:Exit"<<endl<<"\tEnter
choice:";
		cin>>ch;

	cout<<endl<<"================================================
================================================================
================"<<endl;
		switch(ch)
		{
		case 1:
```

```
                    //================================Insertion into
tree=============================================================
================
                    cout<<"\tEnte number:";
                    cin>>num;
                    flag=bst.insert(num);
                    if(flag)
                        cout<<"\t***Inserted successfully***"<<endl;
                    else
                        cout<<"\t***Do not enter duplicate numbers***"<<endl;
                    break;
                case 4:
                    //=====================================Traversal
of
tree=============================================================
===
                    if(bst.getRoot()==NULL)
                    {
                        cout<<"\t***Tree is empty****";
                        continue;
                    }

        cout<<"\t\t1:Inorder"<<endl<<"\t\t2:Preorder"<<endl<<"\t\t3:Postorder"<<e
ndl<<"\t\tEnter choice:";
                    cin>>num;
                    //inorder traversal
                    if(num==1)
                    {
                        cout<<"\t\t::Inorder traversal::";
```

```
                bst.inorder(bst.getRoot());
            }
            //preorder traversal
            else if(num==2)
            {
                cout<<"\t\t::Preorder traversal::";
                bst.preorder(bst.getRoot());
            }
            //postorder traversal
            else{
                cout<<"\t\t::Postorder traversal::";
                bst.postorder(bst.getRoot());
            }
            break;
        case 3:
            //================================searhing in tree=========================================
            cout<<"\tEnter number to be search:";
            cin>>num;
            flag=bst.search(bst.getRoot(),num);
            if(flag)
                cout<<"\t"<<num<<" is present is tree"<<endl;
            else
                cout<<"\t"<<num<<" is not present in tree"<<endl;
            break;
        case 2:
```

```
                //==================================deletion in
tree=========================================
                cout<<"\tEnter number to be deleted:";
                cin>>num;
                temp=bst.deleteNode(bst.getRoot(),num);
                 //if deleted successfully, root will be returned
                if(temp==bst.getRoot())
                        cout<<"\t"<<num<<" is deleted"<<endl;
                else
                        cout<<"\t"<<num<<" is not present"<<endl;
                break;
        case 5:
                //==================================depth of
tree=====================================
                cout<<"\tDepth of tree:"<<bst.treeDepth(bst.getRoot());
                break;
        case 6:
                //==============================creation of mirror
image=====================================
                if(bst.getRoot()==NULL){
                        cout<<"\t****Tree is empty****"<<endl;
                        continue;
                }
                bst.mirrorImg(bst.getRoot());
                cout<<"\t***Mirror Image created***";
                break;
        case 7:
```

```
                //==============================create
copy=========================================
                if(bst.getRoot()==NULL)
                {
                        cout<<"\t***Tree is empty***"<<endl;
                        continue;
                }
                cpy=bst.createCopy(bst.getRoot());
                cout<<"\t***Copy is created***";
                break;
        case 8:
                //============================display parent node
with child=========================
                if(bst.getRoot()==NULL)
                {
                        cout<<"\t***Tree is empty***"<<endl;
                        continue;
                }
                bst.printParentChild(bst.getRoot());
                break;
        case 9:
                //==================Print leaf
nodes=================================================
                cout<<"\tLeaf Nodes:";
                bst.printLeafNode(bst.getRoot());
                break;
        case 10:
```

```
                    //=======================display tree level
wise=============================================
                    if(bst.getRoot()==NULL){
                        cout<<"\t***Empty tree"<<endl;
                        continue;
                    }
                    bst.printLevelWise();
                    break;
                case 0:
                    cout<<"\tThank you"<<endl;
                    break;
                default:
                    cout<<"\t***Invalid choice.....***"<<endl;
                }
        }while(ch!=0);
        return 0;
}
```

- **Output:**

```
Binary Search Tree
================================================================================
1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:1

================================================================================
Ente number:50
***Inserted successfully***

================================================================================
1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:1

================================================================================
```

Ente number:34
Inserted successfully

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:1

==========
==========

Ente number:89
Inserted successfully

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:1

==========
==========

Ente number:1
Inserted successfully

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:26

==========
==========

Invalid choice.....

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:1

==========
==========

Ente number:37

Inserted successfully

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:4

==========
==========

1:Inorder
2:Preorder
3:Postorder
Enter choice:1
::Inorder traversal::1 || 34 || 37 || 50 || 89 ||

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:10

```
================================================================
================================================================
50
34 89
1 37
================================================================
================================================================
        1:Insert node into tree
        2:Delete
        3:Searching
        4:Traversal
        5:Depth of tree
        6:Create mirror image
        7:Create copy
        8:Display all perent nodes with child
        9:Display leaf nodes
        10:Display tree level wise
        0:Exit
        Enter choice:2

================================================================
================================================================
        Enter number to be deleted:1
        1 is deleted

================================================================
================================================================
        1:Insert node into tree
        2:Delete
        3:Searching
        4:Traversal
        5:Depth of tree
        6:Create mirror image
        7:Create copy
        8:Display all perent nodes with child
        9:Display leaf nodes
        10:Display tree level wise
        0:Exit
        Enter choice:4
```

```
==========================================================
==========================================================
        1:Inorder
        2:Preorder
        3:Postorder
        Enter choice:1
        ::Inorder traversal::34 || 37 || 50 || 89 ||
==========================================================
==========================================================
    1:Insert node into tree
    2:Delete
    3:Searching
    4:Traversal
    5:Depth of tree
    6:Create mirror image
    7:Create copy
    8:Display all perent nodes with child
    9:Display leaf nodes
    10:Display tree level wise
    0:Exit
    Enter choice:1

==========================================================
==========================================================
    Ente number:56
    ***Inserted successfully***

==========================================================
==========================================================
    1:Insert node into tree
    2:Delete
    3:Searching
    4:Traversal
    5:Depth of tree
    6:Create mirror image
    7:Create copy
    8:Display all perent nodes with child
    9:Display leaf nodes
    10:Display tree level wise
    0:Exit
```

Enter choice:1

==========
==========

Ente number:98
Inserted successfully

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:4

==========
==========

1:Inorder
2:Preorder
3:Postorder
Enter choice:2
::Preorder traversal::50 || 34 || 37 || 89 || 56 || 98 ||
==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes

10:Display tree level wise
0:Exit
Enter choice:2

==========
==========
Enter number to be deleted:89
89 is deleted

==========
==========
1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:4

==========
==========
1:Inorder
2:Preorder
3:Postorder
Enter choice:1
::Inorder traversal::34 || 37 || 50 || 56 || 98 ||
==========
==========
1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image

7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:10

==========
==========
50
34 98
37 56
==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:2

==========
==========

Enter number to be deleted:34
34 is deleted

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image

7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:4

==========
==========

1:Inorder
2:Preorder
3:Postorder
Enter choice:3
::Postorder traversal::37 || 56 || 98 || 50 ||

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:5

==========
==========

Depth of tree:3

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image

7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:3

==========
==========

Enter number to be search:505
505 is not present in tree

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:3

==========
==========

Enter number to be search:56
56 is present is tree

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image

7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:7

==========
==========
Copy is created
==========
==========
1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:6

==========
==========
Mirror Image created
==========
==========
1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise

0:Exit
Enter choice:10

==========
==========
50
98 37
56
==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:6

==========
==========

Mirror Image created

==========
==========

1:Insert node into tree
2:Delete
3:Searching
4:Traversal
5:Depth of tree
6:Create mirror image
7:Create copy
8:Display all perent nodes with child
9:Display leaf nodes
10:Display tree level wise
0:Exit
Enter choice:8

```
================================================================
================================================================
        Parent:50Child:37 98
        Parent:98Child:56
================================================================
================================================================
        1:Insert node into tree
        2:Delete
        3:Searching
        4:Traversal
        5:Depth of tree
        6:Create mirror image
        7:Create copy
        8:Display all perent nodes with child
        9:Display leaf nodes
        10:Display tree level wise
        0:Exit
        Enter choice:9
================================================================
================================================================
        Leaf Nodes:37 56
================================================================
================================================================
        1:Insert node into tree
        2:Delete
        3:Searching
        4:Traversal
        5:Depth of tree
        6:Create mirror image
        7:Create copy
        8:Display all perent nodes with child
        9:Display leaf nodes
        10:Display tree level wise
        0:Exit
        Enter choice:0

================================================================
================================================================
        Thank you
```