Binary_Search_Tree.cpp

```
/* Roll no.  :  2002
   Batch    :   E-10
*/


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

int main()
{
	node * s;
	BST b;

	do
		{
			int ch;
			cout<<"\tMENU\n";

			cout<<"\t\t1.Create\n\t\t2.Delete\n\t\t3.Search\n\t\t4.Mirror Image"
				"\n\t\t5.Display\n\t\t6.Display Level Wise\n\t\t7.Height of
Tree\n\t\t8.Exit\n\n";
			cout<<"Enter Choice : ";
			cin>>ch;
			cout<<endl;

			switch(ch)
			{
			case 1:
				{
					int n;
					cout<<"Enter number of values you want to enter in
BST : ";
					cin>>n;

					for(int i=0;i<n;i++)
					{
						cout<<"Enter Element number "<<i+1<<" : ";
						int x;
						cin>>x;

						b.create(x);
					}
					cout<<"Binary Search tree created !!!!\n\n";
					break;
				}

			case 2:
```

```
                {
                        int x;
                        cout<<"Enter node you want to delete : ";
                        cin>>x;

                        b.root=b.Delete(b.root,x);
                        cout<<"\n\n";
                        break;
                }

        case 3:
                {
                        int x;
                        cout<<"Enter node you want to search : ";
                        cin>>x;
                        s=b.search(b.root,x);
                        cout<<"\n\n";
                        break;
                }

        case 4:
                {
                        node *m;
                        m=b.mirror(b.root);
                        cout<<"Mirror Image created \n\n";

                        cout<<"Created tree in Inorder is : ";
                        b.display(m);
                        cout<<"\n\n";
                        break;
                }

        case 5:
                {
                        cout<<"Created tree in Inorder is : ";
                        b.display(b.root);
                        cout<<"\n\n";
                        break;
                }

        case 6:
                {
                        cout<<"Created tree in Level order is : ";
                        b.display_level(b.root);
                        cout<<"\n\n";
                        break;
                }

        case 7:
                {
                        cout<<"Height of tree is :
"<<b.height(b.root)<<endl<<endl;
```

```
                                        break;
                                }

                        case 8:
                                {
                                        exit(0);
                                        break;
                                }
                        }
                }
                while(1);

        return 0;
}
```

BST.cpp

```
/*
 * BST.cpp
 *
 *  Created on: 12-Feb-2018
 *      Author: e2002
 */

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

BST::BST()
{
	root=NULL;
}

node * getnode(int x)
{
	node * n;
	n=new node;

	n->data=x;
	n->right=NULL;
	n->left=NULL;

	return n;
}

void BST:: create(int val)
{
	if(root==NULL)
	{
		root=getnode(val);
		return;
	}

	node *p,*q;

	p=q=root;

	while(q!=NULL && q->data!=val)
	{
		p=q;

		if(p->data > val)
			q=p->left;

		else
			q=p->right;
```

```
        }

        if(val == p->data)
        {
                cout<<"Duplicate item !!!"<<endl<<endl;
                return;
        }

        else if(val< p->data)
                p->left=getnode(val);

        else
        {
                p->right=getnode(val);
        }

}

node*  BST:: search(node *root, int val)
{
        if(root==NULL)
                return NULL;

        if(val == root->data)
        {
                cout<<"Found!!!!"<<endl;
                cout<<""<<root->data<<" : \n";
                return root;
        }

        if(val > root->data)
                return search(root->right,val);

        else
                return search(root->left,val);
}

int BST :: height(node * root)
{
        if(root==NULL)
                return 0;

        int left_height=0,right_height=0;

        left_height =height(root->left)+1;
        right_height=height(root->right)+1;

        return (left_height>right_height) ?  left_height :  right_height;
}

node* BST :: mirror(node * root)
{
```

```
        if(root != NULL)
        {
                node* temp;

                temp=root->left;
                root->left=root->right;
                root->right=temp;

                 mirror(root->left);
                 mirror(root->right);
        }

        return root;
}

void BST :: display(node * root)
{
        if(root==NULL)
                return;

        display(root->left);
        cout<<root->data<<" ";
        display(root->right);
}

void BST :: display_level(node * root)
{
        queue<node *> q1, q2;

    if (root == NULL)
       return;

    q1.push(root);

    while (!q1.empty() || !q2.empty())
    {
       while (!q1.empty())
       {
          if (q1.front()->left != NULL)
             q2.push(q1.front()->left);

          if (q1.front()->right != NULL)
             q2.push(q1.front()->right);

          cout << q1.front()->data << " ";
          q1.pop();
       }

       cout << "\n";

       while (!q2.empty())
       {
```

```
            if (q2.front()->left != NULL)
                q1.push(q2.front()->left);

            if (q2.front()->right != NULL)
                q1.push(q2.front()->right);

            cout << q2.front()->data << " ";
            q2.pop();
        }

        cout << "\n";
    }
}

int find_left_most(node * n)
{
        if(n->left==NULL)
                return n->data;

        else
                return find_left_most(n->left);
}

node * delete_left_most(node *n)
{
        if(n->left==NULL)
                return n->right;

        else
        {
                n->left=delete_left_most(n->left);
                return n;
        }
}

node * deletenode(node * n)
{

        if(n->left==NULL && n->right==NULL)  // leaf
                return NULL;

        if(n->left ==NULL)                                      // only right child
                return n->right;

        if(n->right == NULL)                            // only left child
                return n->left;

        n->data = find_left_most(n->right);

        n->right= delete_left_most(n->right);
        return n;
}
```

```
node* BST :: Delete(node * root,int key)
{
	node * p;

	if(root==NULL)
	{
		cout<<"Not Found !!!\n\n";
		return NULL;
	}

	else
	{
		if(key == root->data)
				root=deletenode(root);

		else if(key < root->data)
			root->left=Delete(root->left,key);

		else
			root->right=Delete(root->right,key);

		return root;
	}
-
}
```

BST.h

```
/*
 * BST.h
 *
 *  Created on: 12-Feb-2018
 *      Author: e2002
 */

#ifndef BST_H_
#define BST_H_

struct node
{
	int data;
	node * right;
	node * left;
};

class BST
{
public :

	BST();
	node * root;
	void create(int val);
	node* search(node *,int val);
	node* mirror(node *);
	void display_level(node *);
	void display(node *);
	int height(node *);
	node* Delete(node *,int);
};

#endif /* BST_H_ */
```