JASKIRAT SINGH
2020CSC1008
BST

Code

```
#include <iostream>
#include <stack>
#include <stdlib.h>
#include <time.h>
using namespace std;
class Node
{
public:
    int data;
    Node *left, *right;
    Node() // Default constructor.
    {
        data = 0;
        left = right = NULL;
    }
    Node(int value) // Parameterized constructor.
    {
        data = value;
        left = right = NULL;
```

```
    }
};

class BST
{
public:
    Node *root;
    int count;

    BST()
    {
        root = NULL;
        count = 0;
    }
    Node *createNode(int data);
    void Insert(int data);
    Node *search(int data);
    void inorder(Node *troot);
    void preorder(Node *troot);
    void postorder(Node *troot);
    void iterativeInorder();
    void iterativePreorder();
    void iterativePostorder();
    void findAndDeleteByMerging(int el);
    void findAndDeleteByCopying(int el);
    void deleteByMerging(Node *&node);
    void deleteByCopying(Node *&node);
};

Node *BST::createNode(int data)
{
    Node *newNode = new Node;
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void BST ::Insert(int value)
{
    Node *newNode = createNode(value);
    if (!root)
    {
        root = newNode; // Insert the first node, if root is NULL.
```

```
        count++;
        return;
    }
    Node *temp = root, *prev = NULL;
    while (temp != NULL)
    { // find a place for inserting new node;
        prev = temp;
        if (value < temp->data)
            temp = temp->left;
        else
            temp = temp->right;
    }
    if (value < prev->data)
        prev->left = newNode;
    else
        prev->right = newNode;
    count++;
}

// complexity is the length of the path leading to this node plus 1
Node *BST::search(int el)
{
    Node *p = root;
    while (p != NULL)
    {
        if (el == p->data)
            return p;
        else if (el < p->data)
            p = p->left;
        else
            p = p->right;
    }

    return NULL;
}

void BST::inorder(Node *troot)
{
    if (troot != NULL)
    {
        inorder(troot->left);
        cout << troot->data << " ";
        inorder(troot->right);
    }
```

```
}
void BST::preorder(Node *troot)
{
    if (troot != NULL)
    {
        cout << troot->data << " ";
        preorder(troot->left);
        preorder(troot->right);
    }
}
void BST::postorder(Node *troot)
{
    if (troot != NULL)
    {
        postorder(troot->left);
        postorder(troot->right);
        cout << troot->data << " ";
    }
}

void BST::iterativeInorder()
{
    stack<Node *> travStack;
    Node *p = root;
    while (p != NULL)
    {
        while (p != NULL)
        { // stack the right child (if any) and the node itself when going
            if (p->right)
                travStack.push(p->right); // to the left;
            travStack.push(p);
            p = p->left;
        }
        p = travStack.top();
        travStack.pop();
        while (!travStack.empty() && p->right == NULL)
        { // visit it and all nodes with no right child
            cout << p->data << " ";
            p = travStack.top();
            travStack.pop();
        }
        cout << p->data << " "; // visit also the first node with a right child (if any)
        if (!travStack.empty())
        {
```

```
            p = travStack.top();
            travStack.pop();
        }
        else
            p = NULL;
    }
}

void BST::iterativePreorder()
{
    stack<Node *> travStack;
    Node *p = root;
    while (p != NULL)
    {
        do
        {
            cout << p->data << " ";
            if (p->right != NULL)
                travStack.push(p->right);
            if (p->left != NULL) // left child pushed after right to be on the top
                travStack.push(p->left);
            p = travStack.top();
            travStack.pop();
        } while (!travStack.empty());
    }
}

void BST::iterativePostorder()
{
    stack<Node *> travStack;
    Node *p = root, *q = root;
    while (p != NULL)
    {
        while (p->left != NULL)
        {
            travStack.push(p);
            p = p->left;
        }
        while (p->right == NULL || p->right == q)
        {
            cout << p->data << " ";
            q = p;
            if (travStack.empty())
                return;
```

```
            p = travStack.top();
            travStack.pop();
        }
        travStack.push(p);
        p = p->right;
    }
}

void BST::findAndDeleteByMerging(int el)
{
    Node *node = root, *prev = NULL;
    while (node != NULL)
    {
        if (node->data == el)
            break;
        prev = node;
        if (el < node->data)
            node = node->left;
        else
            node = node->right;
    }
    if (node != NULL && node->data == el)
    {
        if (node == root)
            deleteByMerging(root);
        else if (prev->left == node)
            deleteByMerging(prev->left);
        else
            deleteByMerging(prev->right);
    }
    else if (root != NULL)
        cout << "element" << el << "is not in the tree\n";
    else
        cout << "the tree is empty\n";
}

void BST::deleteByMerging(Node *&node)
{
    Node *tmp = node;
    if (node != NULL)
    {
        if (!node->right)           // node has no right child:
            node = node->left;      // its left child (if any) is attached to its parent;
        else if (node->left == NULL) // node has no left child:
```

```
            node = node->right;      //its right child is attached to its parent;
        else
        {                            // merge subtrees;
            tmp = node->left;          // 1. move left
            while (tmp->right != NULL) // 2. and then right as far as possible;
                tmp = tmp->right;
            tmp->right = node->right; // link the rightmost node of the left subtree and
            tmp = node;               // the right subtree;
            node = node->left;
        }
        delete tmp;
    } //end of if
}

void BST::findAndDeleteByCopying(int el)
{
    Node *node = root, *prev = NULL;
    while (node != NULL)
    {
        if (node->data == el)
            break;
        prev = node;
        if (el < node->data)
            node = node->left;
        else
            node = node->right;
    }
    if (node != NULL && node->data == el)
    {
        if (node == root)
            deleteByCopying(root);
        else if (prev->left == node)
            deleteByCopying(prev->left);
        else
            deleteByCopying(prev->right);
    }
    else if (root != NULL)
        cout << "element" << el << "is not in the tree\n";
    else
        cout << "the tree is empty\n";
}

void BST::deleteByCopying(Node *&node)
{
```

```
    Node *previous, *tmp = node;
    if (node->right == NULL) // node has no right child;
        node = node->left;
    else if (node->left == NULL) // node has no left child;
        node = node->right;
    else
    {
        tmp = node->left; // node has both children;
        previous = node;  // 1.
        while (tmp->right != 0)
        { // 2.
            previous = tmp;
            tmp = tmp->right;
        }
        node->data = tmp->data; // 3.
        if (previous == node)
            previous->left = tmp->left;
        else
            previous->right = tmp->left; // 4.
    }
    delete tmp; // 5.
}

int main()
{
    BST bst;

    int choice;
    char ans;

    do
    {

        cout << "\n*********Menu**********"
             << "\n1. Insert"
             << "\n2. Search"
             << "\n3. Recursive Inorder Traversal"
             << "\n4. Recursive Preorder Traversal"
             << "\n5. Recursive Postorder Traversal"
             << "\n6. Iterative Inorder Traversal"
             << "\n7. Iterative Preorder Traversal"
             << "\n8. Iterative Postorder Traversal"
             << "\n9. Delete By Merging"
             << "\n10. Delete By Copying"
```

```
     << "\n11. exit \n\n"
     << "Enter your choice... ";
cin >> choice;

switch (choice)
{
case 1:
{
    cout << "\n Enter the number of nodes... ";
    int n;
    cin >> n;

    cout << "\n Inserting " << n << " random numbers...\n";
    srand(time(0));
    for (int i = 0; i < n; i++)
    {
        int random = rand() % 50;
        cout << "\nInserting " << random << "...";
        bst.Insert(random);
    }
    cout << "\n"
         << n << " random numbers inserted successfully!";

    break;
}

case 2:
{
    cout << "\nEnter the element to be searched.. ";
    int ele;
    cin >> ele;

    cout << "\nElement present at address : " << bst.search(ele);

    break;
}

case 3:
{
    Node *head = new Node;
    head = bst.root;
    cout << "\nElements in the tree : ";
    bst.inorder(head);
    break;
```

```
}

case 4:
{
    Node *head = new Node;
    head = bst.root;
    cout << "\nElements in the tree : ";
    bst.preorder(head);
    break;
}

case 5:
{
    Node *head = new Node;
    head = bst.root;
    cout << "\nElements in the tree : ";
    bst.postorder(head);
    break;
}

case 6:
{
    cout << "\nElements in the tree : ";
    bst.iterativeInorder();
    break;
}

case 7:
{
    cout << "\nElements in the tree : ";
    bst.iterativePreorder();
    break;
}

case 8:
{
    cout << "\nElements in the tree : ";
    bst.iterativePostorder();
    break;
}

case 9:
{
    cout << "\nEnter the element to be deleted... ";
```

```
            int ele;
            cin >> ele;

            bst.findAndDeleteByMerging(ele);
            cout << "\nDeleted successfully!";
            break;
        }

        case 10:
        {
            cout << "\nEnter the element to be deleted... ";
            int ele;
            cin >> ele;

            bst.findAndDeleteByCopying(ele);
            cout << "\nDeleted successfully!";
            break;
        }

        default:
        {
            exit(101);
        }
        }

        cout << "\n\nDo you want to continue?"
             << "\nPress y to continue"
             << "\nelse press n... ";
        cin >> ans;
    } while (ans == 'y');

    return 0;
}
```

Output

```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 1

 Enter the number of nodes... 6

 Inserting 6 random numbers...

Inserting 40...
Inserting 6...
Inserting 23...
Inserting 23...
Inserting 38...
Inserting 32...
6 random numbers inserted successfully!

Do you want to continue?
Press y to continue
else press n...
```

```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 2

Enter the element to be searched.. 23

Element present at address : 0x6c5b30

Do you want to continue?
Press y to continue
else press n...
```

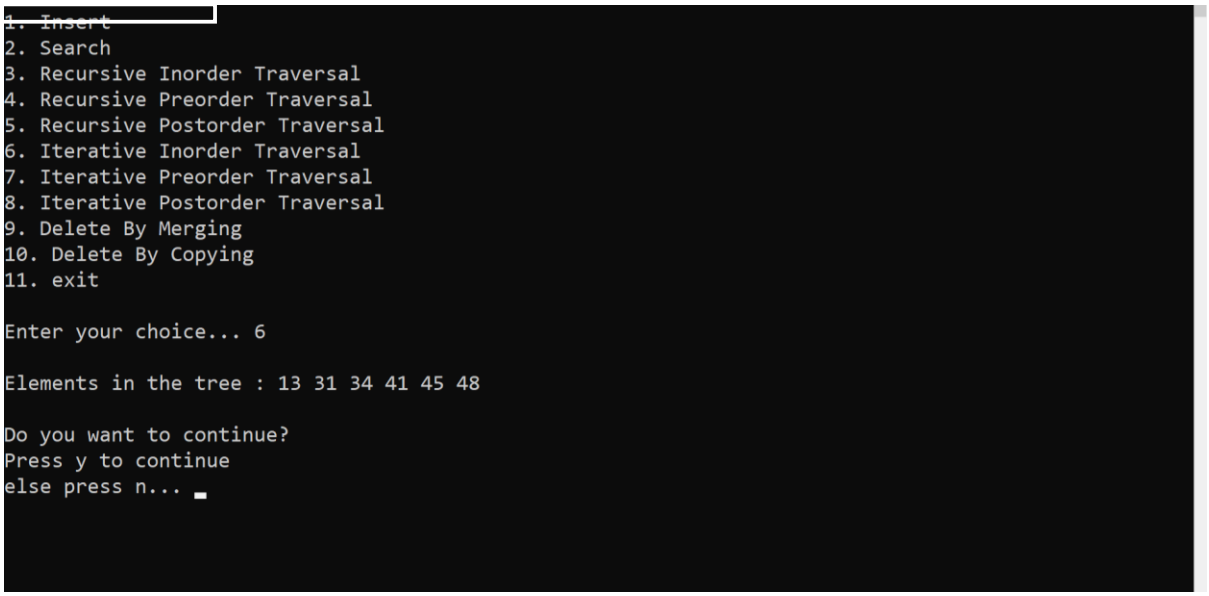
```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 9

Enter the element to be deleted... 23

Deleted successfully!

Do you want to continue?
Press y to continue
else press n...
```

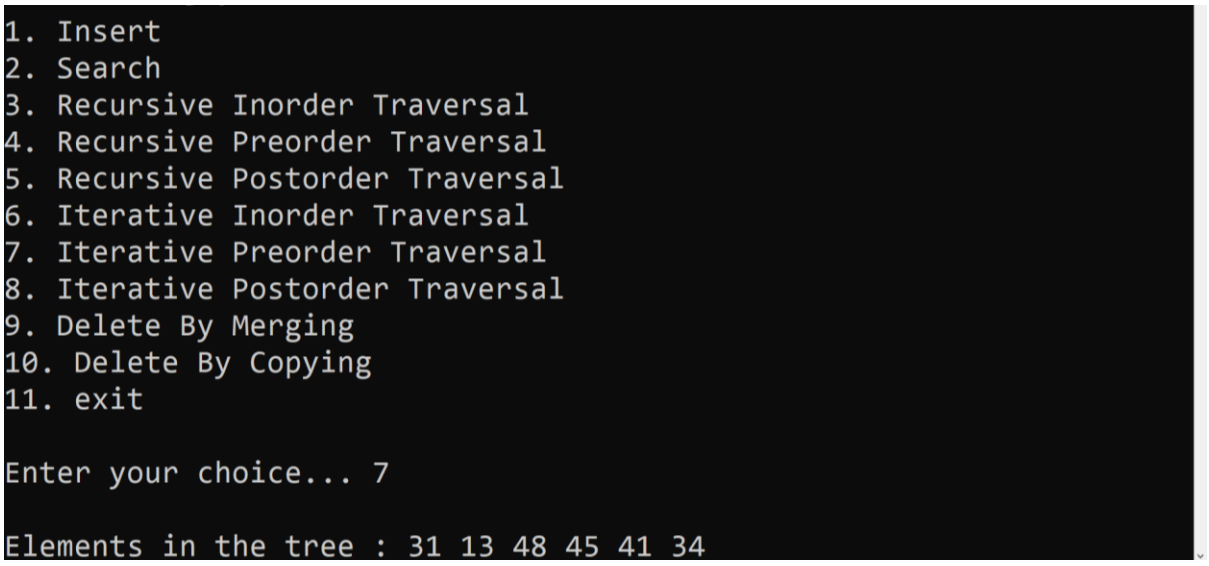
```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 10

Enter the element to be deleted... 33

Deleted successfully!

Do you want to continue?
Press y to continue
else press n...
```

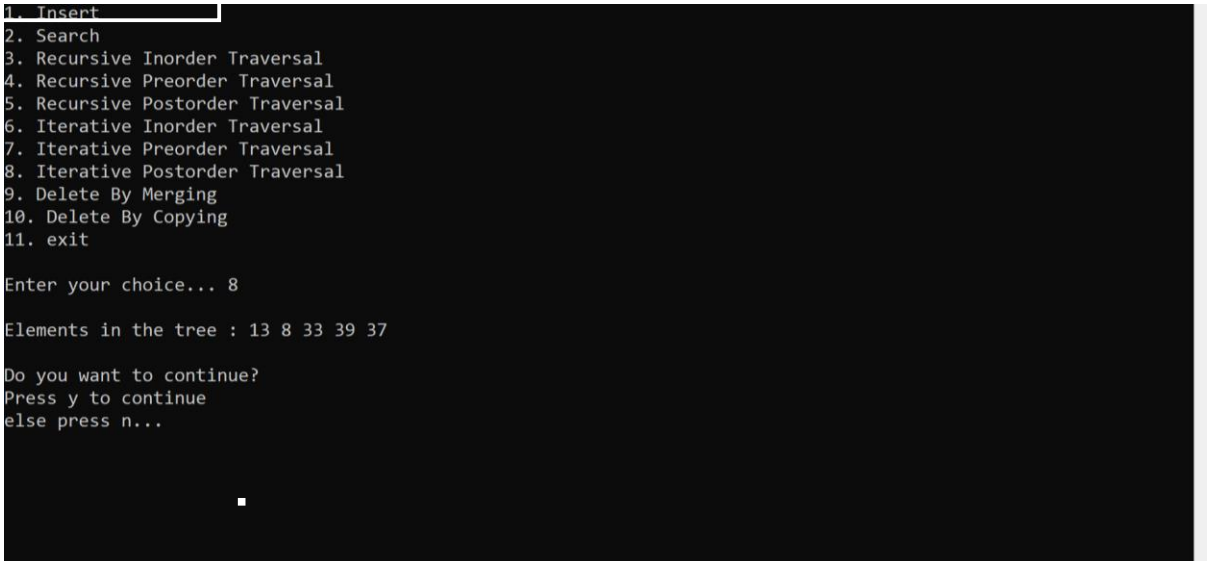
```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 3

Elements in the tree : 13 31 34 41 45 48

Do you want to continue?
Press y to continue
else press n...
```

```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 4

Elements in the tree : 31 13 48 45 41 34

Do you want to continue?
Press y to continue
else press n...
```

```
1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit

Enter your choice... 5

Elements in the tree : 13 34 41 45 48 31

Do you want to continue?
Press y to continue
else press n...
```


1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit
Enter your choice... 6
Elements in the tree : 13 31 34 41 45 48
Do you want to continue?
Press y to continue
else press n...


1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit
Enter your choice... 7
Elements in the tree : 31 13 48 45 41 34


1. Insert
2. Search
3. Recursive Inorder Traversal
4. Recursive Preorder Traversal
5. Recursive Postorder Traversal
6. Iterative Inorder Traversal
7. Iterative Preorder Traversal
8. Iterative Postorder Traversal
9. Delete By Merging
10. Delete By Copying
11. exit
Enter your choice... 8
Elements in the tree : 13 8 33 39 37
Do you want to continue?
Press y to continue
else press n...