```cpp
// Question1.cpp : This file contains the 'main' function. Program execution begins and ends there.
////////// /
//-------------------------------------------------------------------------------------------
//Name                          Sai Chaitanya Kilambi
//Course                        CPSC 131 Data Structures, Fall, 2022
//Assignment                    No.11 question:1
//Due date                      11/30/2022
// Purpose:
// This program demonstrates insertion of data in array Days into a Binary Search Tree along with preorder,inorder
// and postorder traversal. It also demonstrates how to search display and count the number of leaves
//-------------------------------------------------------------------------------------------
// list of libraries
//
//importing the required libraries

#include <iostream>
using namespace std;

//Node class
class Node {
public:
    string data;
    Node* left;
    Node* right;
    //constructor
    Node(string data) {
        this->data = data;
        left = NULL;
        right = NULL;
    }
};

//BST class modified
class BST
{
    //root of the tree as attribute.
private:
    Node* root;


public:
    //constructors
    BST() {
        root = NULL;
```

```
44      }
45      BST(string data) {
46          root = new Node(data);
47      }
48
49
50      //one private helper method for each utility function.
51  private:
52      //insert helper function
53      void insert(Node*& root, string data) {
54          if (root == NULL) {
55              root = new Node(data);
56          }
57          else
58          {
59              if (root->data > data) {
60                  insert(root->left, data);
61              }
62              else {
63                  insert(root->right, data);
64              }
65          }
66      }
67  public:
68      //insert function
69      void insert(string data) {
70          insert(root, data);
71      }
72
73
74  private:
75      //inorder helper function
76      void inorder(Node* curr) {
77          if (curr == NULL) {
78              return;
79          }
80
81          inorder(curr->left);
82          cout << curr->data << " ";
83          inorder(curr->right);
84      }
85  public:
86      //inorder function
87      void inorder() {
88          cout << "Inorder traversal is :- ";
89          inorder(root);
90          cout << endl;
91      }
92
```

```
 93
 94  private:
 95      //postorder helper function
 96      void postorder(Node* curr) {
 97          if (curr == NULL) {
 98              return;
 99          }
100
101          inorder(curr->left);
102          inorder(curr->right);
103          cout << curr->data << " ";
104      }
105  public:
106      //postorder function
107      void postorder() {
108          cout << "Postorder traversal is:- ";
109          postorder(root);
110          cout << endl;
111      }
112
113
114  private:
115      //preorder helper function
116      void preorder(Node* curr) {
117          if (curr == NULL) {
118              return;
119          }
120
121          cout << curr->data << " ";
122          inorder(curr->left);
123          inorder(curr->right);
124      }
125  public:
126      //preorder function
127      void preorder() {
128          cout << "Preorder traversal is:- ";
129          preorder(root);
130          cout << endl;
131      }
132
133
134  private:
135      //display leaves helper function
136      void displayleaves(Node* curr) {
137          if (curr == NULL) {
138              return;
139          }
140          if (curr->left == NULL && curr->right == NULL) {
141              cout << curr->data << " ";
```

```
142             return;
143         }
144         displayleaves(curr->left);
145         displayleaves(curr->right);
146     }
147 public:
148     //display leaves function
149     void displayleaves() {
150         cout << "Only leaves of tree are:- ";
151         displayleaves(root);
152         cout << endl;
153     }
154 
155 
156 private:
157     //display nodes with one child helper function
158     void Nodewithonechild(Node* curr) {
159         if (curr == NULL)
160             return;
161         if ((curr->left == NULL) ^ (curr->right == NULL)) {
162             cout << curr->data << " ";
163         }
164         Nodewithonechild(curr->left);
165         Nodewithonechild(curr->right);
166     }
167 public:
168     //display nodes with one child function
169     void Nodewithonechild() {
170         cout << "Nodes with one child are:- ";
171         Nodewithonechild(root);
172         cout << endl;
173     }
174 
175 
176 private:
177     //height helper function
178     int height(Node* curr) {
179         if (curr == 0)
180             return 0;
181         return 1 + max(height(curr->left), height(curr->right));
182     }
183 public:
184     //height function
185     int height() {
186         return height(root);
187     }
188 
189 private:
190     //search helper function
```

```cpp
    void search(Node* curr, string data) {
        if (curr == NULL) {
            cout << data << " not found" << endl;
            return;
        }

        if (curr->data == data) {
            cout << data << " found" << endl;
            return;
        }

        if (curr->data > data) {
            search(curr->left, data);
        }
        else {
            search(curr->right, data);
        }
    }
public:
    //search function
    void search(string data) {
        cout << "Searching " << data << ":- ";
        search(root, data);
    }

private:
    //number of nodes helper function
    int NumberofNodes(Node* curr) {
        if (curr == 0)
            return 0;
        return 1 + NumberofNodes(curr->left) + NumberofNodes(curr->right);
    }
public:
    //number of nodes function
    int NumberofNodes() {
        return NumberofNodes(root);
    }
};

//main
int main() {
    //given array
    string Days[7] = { "MON","TUE","WED","THR","FRI","SAT","SUN" };

    BST* tree = new BST();
    for (string day : Days)
        tree->insert(day);

    tree->inorder();
```

```
240     tree->postorder();
241     tree->preorder();
242     tree->displayleaves();
243     tree->Nodewithonechild();
244     int height = tree->height();
245     cout << "Height of BST is " << height << endl;
246     //searching for Mon
247     tree->search("Mon");
248     //searching for THR
249     tree->search("THR");
250     cout << "Number of nodes in BST is " << tree->NumberofNodes() << endl;
251 }
```