Name: Alcantara, Jason P.
Course & Section: CPE 010 - CPE21S4

1. Define what is a binary tree?

- A binary is a type of data structure where each node can have at most two children but it's usually called the left and right child. It's used to organize data in a hierarchical way. The top node of the tree is called root, and nodes without any children are called leaves. They provide a flexible and most efficient way to store data information.

2. What are the key components of a Binary tree?

- The main parts of a tree are the nodes which hold the actual data. Each node also has two pointers or links one to its left child and one to the right child. The very first node of the tree is called the root. If a node doesn't have any children, then it's called a leaf node. These basic components together form the structure of the tree and allow it to grow and be guided easily.

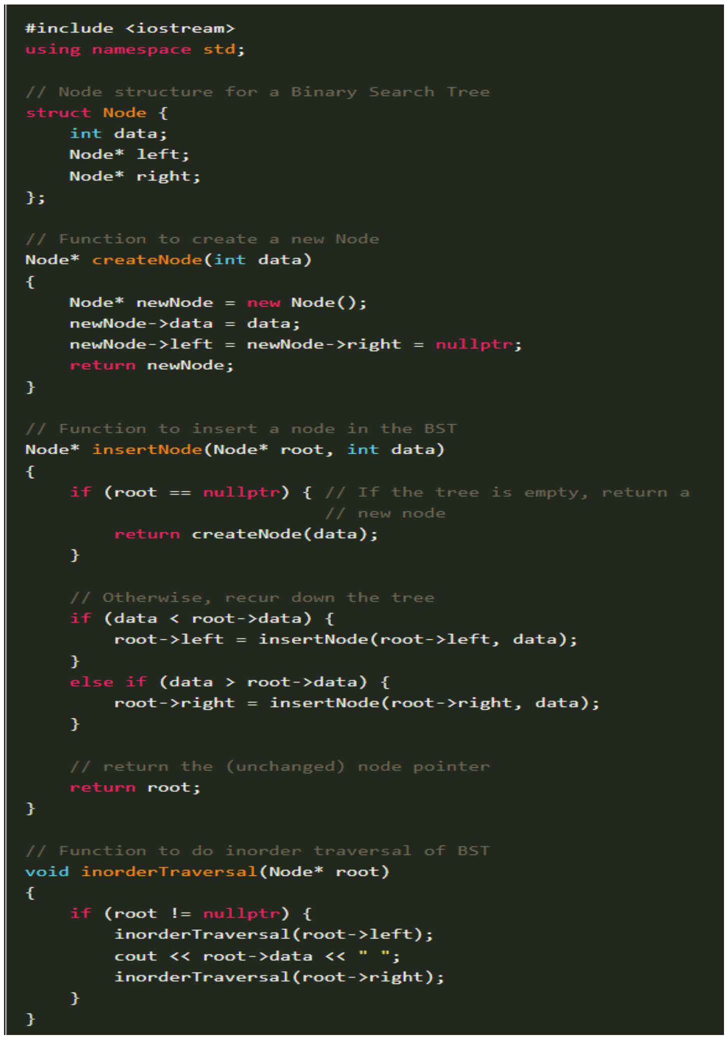
3. What are the operations that can be performed in a Binary tree?

- There are several operations that can be used on binary tree like inserting the new nodes or deleting the existing ones. You can also search for specific values in the. One important operation is traversal, which means visiting all the nodes in a certain order like in order, pre order or post order. The operations can help in

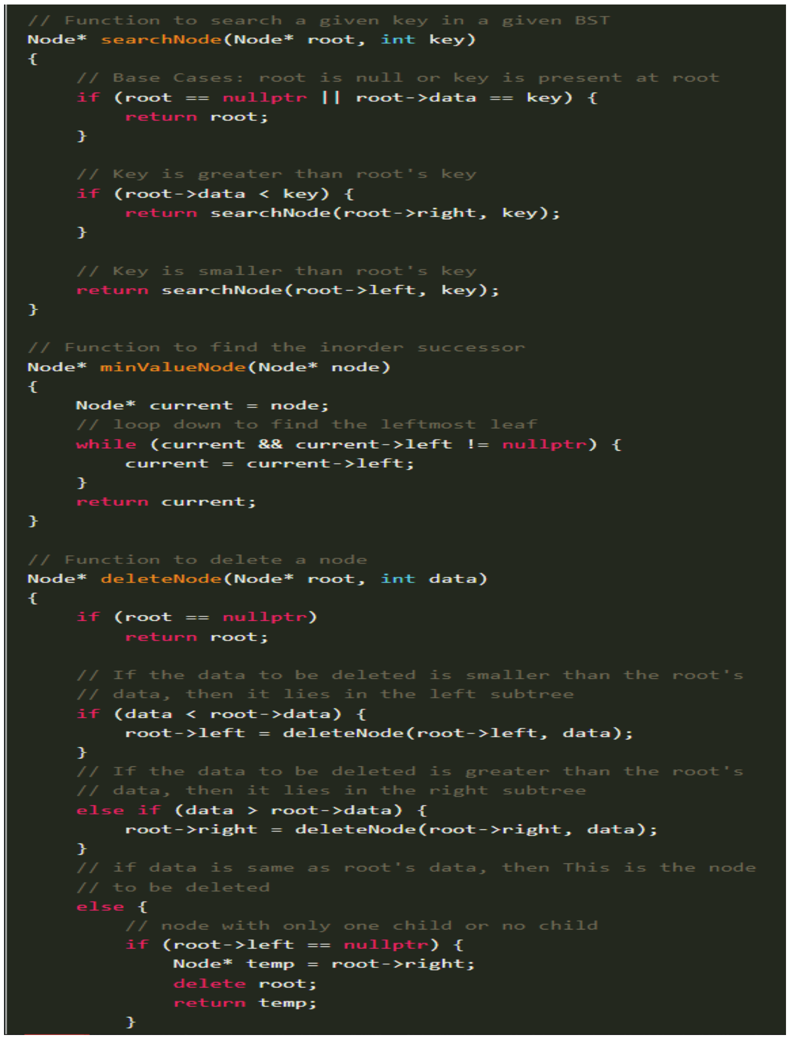
4. What are the conditions for a Binary Search Tree?

- A binary search Tree (BST) is a special kind of binary tree that follows a specific rule for any node: all values in the left subtree must be less than the node’s value and all values in the right subtree must be greater. This rule keeps the tree organized for fast searching. The BST property applies to every node in the tree, not just the root. Because of this structure searching, inserting and deleting can be done more efficiently than in an unsorted tree. It's one of the most useful types of binary tree in coding or programming

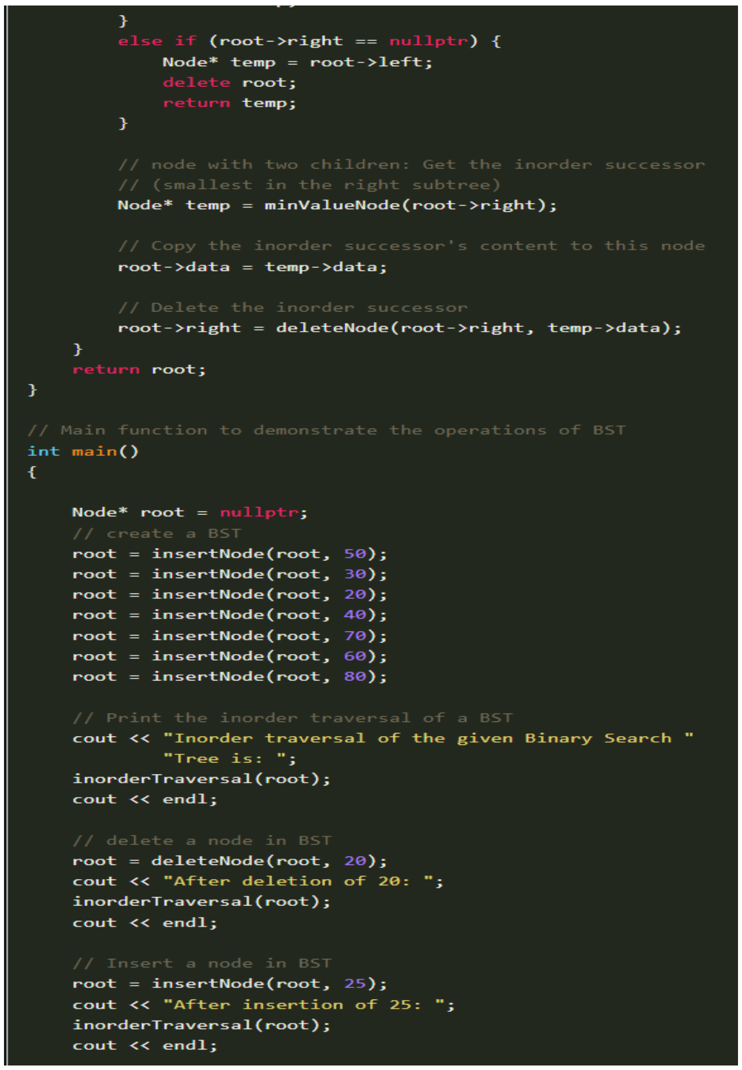
5. Give a simple sample program in C++ that uses the above algorithm. Explain how the program works.



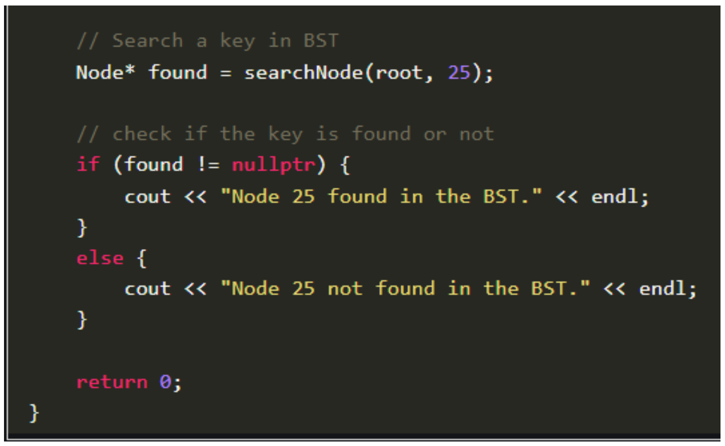
```cpp
#include <iostream>
using namespace std;

// Node structure for a Binary Search Tree
struct Node {
    int data;
    Node* left;
    Node* right;
};

// Function to create a new Node
Node* createNode(int data)
{
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

// Function to insert a node in the BST
Node* insertNode(Node* root, int data)
{
    if (root == nullptr) { // If the tree is empty, return a
                           // new node
        return createNode(data);
    }

    // Otherwise, recur down the tree
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    }
    else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }

    // return the (unchanged) node pointer
    return root;
}

// Function to do inorder traversal of BST
void inorderTraversal(Node* root)
{
    if (root != nullptr) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}
```



```cpp
// Function to search a given key in a given BST
Node* searchNode(Node* root, int key)
{
    // Base Cases: root is null or key is present at root
    if (root == nullptr || root->data == key) {
        return root;
    }

    // Key is greater than root's key
    if (root->data < key) {
        return searchNode(root->right, key);
    }

    // Key is smaller than root's key
    return searchNode(root->left, key);
}

// Function to find the inorder successor
Node* minValueNode(Node* node)
{
    Node* current = node;
    // loop down to find the leftmost leaf
    while (current && current->left != nullptr) {
        current = current->left;
    }
    return current;
}

// Function to delete a node
Node* deleteNode(Node* root, int data)
{
    if (root == nullptr)
        return root;

    // If the data to be deleted is smaller than the root's
    // data, then it lies in the left subtree
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    }
    // If the data to be deleted is greater than the root's
    // data, then it lies in the right subtree
    else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    }
    // if data is same as root's data, then This is the node
    // to be deleted
    else {
        // node with only one child or no child
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
```



```cpp
        }
        else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        // node with two children: Get the inorder successor
        // (smallest in the right subtree)
        Node* temp = minValueNode(root->right);

        // Copy the inorder successor's content to this node
        root->data = temp->data;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Main function to demonstrate the operations of BST
int main()
{

    Node* root = nullptr;
    // create a BST
    root = insertNode(root, 50);
    root = insertNode(root, 30);
    root = insertNode(root, 20);
    root = insertNode(root, 40);
    root = insertNode(root, 70);
    root = insertNode(root, 60);
    root = insertNode(root, 80);

    // Print the inorder traversal of a BST
    cout << "Inorder traversal of the given Binary Search "
            "Tree is: ";
    inorderTraversal(root);
    cout << endl;

    // delete a node in BST
    root = deleteNode(root, 20);
    cout << "After deletion of 20: ";
    inorderTraversal(root);
    cout << endl;

    // Insert a node in BST
    root = insertNode(root, 25);
    cout << "After insertion of 25: ";
    inorderTraversal(root);
    cout << endl;
```



```cpp
    // Search a key in BST
    Node* found = searchNode(root, 25);

    // check if the key is found or not
    if (found != nullptr) {
        cout << "Node 25 found in the BST." << endl;
    }
    else {
        cout << "Node 25 not found in the BST." << endl;
    }

    return 0;
}
```

Explanations:
In this c++ program it creates and manages a Binary Search Tree (BST) using its basic operations like insert, delete, search and inorder traversal. Each node in the tree stores a number and links to its left and right child nodes. The insertion node function places new values in the correct spot based on BST rules. The program also includes delete node to remove values, search node to clarify if the value does exist and finally inorder traversal it prints the tree in sorted order, showing on how the data structure keeps the data organized.

“I affirm that I have not given or received any unauthorized help on this assignment, and that this work is my own.”