```
#pragma once
#ifndef BINARY_SEARCH_TREE_H
#define BINARY_SEARCH_TREE_H

#include "binaryNode.h";
#include "tree.h";
#include <Windows.h>
#include <iostream>

using namespace std;


class BinarySearchTree : public tree {

	private:
		BinaryNode *head;


	public:

		BinarySearchTree() {
			this->head = nullptr;
		}

		virtual void addNode(long long int iKey, int iRowNumber) {
			if (head == nullptr) {
				head = new BinaryNode(iKey, iRowNumber);
				return;
			}
			BinaryNode* root = head;
			BinaryNode* rootParent = head;

			while (root != nullptr) {
				rootParent = root;
				(iKey < root->iKey) ? root = root->left : root =
root->right;
			}

			(rootParent->left == nullptr && rootParent->iKey > iKey) ?
rootParent->left = new BinaryNode(iKey, iRowNumber) : (rootParent->right ==
nullptr && rootParent->iKey < iKey) ? rootParent->right = new BinaryNode(iKey,
iRowNumber)  : 0;

		}

		virtual int findNode(long long int iKey) {
			BinaryNode* root = head;

			while (root != nullptr && iKey != root->iKey)
			{
				count++;
				(iKey < root->iKey) ? root = root->left : root =
root->right;
			}

			return (root == nullptr) ? -1 : root->iRowNumber;
		}
```

```
		virtual void deleteNode(long long int iKey) {
			BinaryNode* root = head;
			BinaryNode* parent = root;

			while (root != nullptr && iKey != root->iKey)
			{
				parent = root;
				(iKey < root->iKey) ? root = root->left : root =
root->right;
			}

			if (root == nullptr) return;

			if (root->left == nullptr && root->right == nullptr) {
				(parent->left == root) ? parent->left = nullptr :
parent->left = nullptr;

			}
			else if (root->left != nullptr && root->right == nullptr)
{
				root->swap(root->left);
			}
			else if (root->right != nullptr && root->left == nullptr)
{
				root->swap(root->right);

			}
			else {
				BinaryNode* tmp = root->right;
				BinaryNode* tmpParent = root;
				while (tmp != nullptr && tmp->left != nullptr)
				{
					tmpParent = tmp;
					tmp = tmp->left;
				}
				root->ValueSwap(tmp);

				(tmpParent->right == tmp) ? tmpParent->right =
nullptr : tmpParent->left =nullptr;
			}
		}

		virtual void print() {
			printExecute(head, 0);
		}

		void printExecute(Node* root, int level) {
			BinaryNode* rootBinary = (BinaryNode*)root;
			if (rootBinary == nullptr) return;
			printExecute(rootBinary->right, level + 1);
			for (int i = 0; i < level; i++) cout << "\t";
			cout << " " << rootBinary->iKey << "\n";
			printExecute(rootBinary->left, level + 1);
		}


		BinaryNode* getHead() {
			return head;
```

```
            }

            void generateTree(int iSize) {
                  for (int i = 0; i < iSize - 1; i++) {
                        this->addNode(rand() % 100 , i);
                  }
            }
};



#endif // !BINARY_SEARCH_TREE_H
```