HW6 層序走訪

HW6CPP.cpp

```
#include <iostream>
#include <cstring>
#include "CBinarySearchTree.h"
int main()
{
    std::string success = "Success to remove number ";
    std::string fail = "Fail to remove because there is no number ";
    int a[] = { 41, 67, 34, 0, 69, 24, 78, 58, 62, 64,\
        5, 45, 81, 27, 61, 91, 95, 42, 27, 36,\
        91, 4, 2, 53, 92, 82, 21, 16, 18, 95,\
        47, 26, 71, 38, 69, 12, 67, 99, 35, 94,\
        3, 11, 22, 33, 73, 64, 41, 11, 53, 68 };
```

```
    // demo: insertion
    CBinarySearchTree<int> bst;
    for (int i = 0; i < sizeof(a) / sizeof(int); i++)
    {
        std::cout << "Insert: " << a[i] << std::endl;
        if (!bst.Insert(a[i]))
            std::cout << "Insert fail: " << a[i] << std::endl;
    }
    bst.LevelOrder(bst.m_Root);
    std::cout << std::endl;

    // demo: remove inexist number
    std::cout << "After remove 1:" << std::endl;
    if (bst.Remove(1))
        std::cout << success << 1 << " in the tree!\n";
    else
        std::cout << fail << 1 << " in the tree!\n";
    bst.LevelOrder(bst.m_Root);
    std::cout << "\n";
```

```
    // demo: remove the node with only one son
    std::cout << "After remove 2:" << std::endl;
    if (bst.Remove(2))
        std::cout << success << 2 << " in the tree!\n";
    else
        std::cout << fail << 2 << " in the tree!\n";
    bst.LevelOrder(bst.m_Root);
    std::cout << std::endl;

    // demo: remove the node with two sons
    std::cout << "After remove 16:" << std::endl;
    if (bst.Remove(16))
        std::cout << success << 16 << " in the tree!\n";
    else
        std::cout << fail << 16 << " in the tree!\n";
    bst.LevelOrder(bst.m_Root);
    std::cout << std::endl;

    return 0;
}
```

CNode.h

```
#pragma once
template<class T>
class CNode
{
public:
    CNode();
    ~CNode();
    CNode<T>* m_Left;
    CNode<T>* m_Right;
    T m_Value;
    bool m_IsEmpty;
};

template<class T>
inline CNode<T>::CNode()
    : m_Left(NULL)
    , m_Right(NULL)
    , m_IsEmpty(true)
    , m_Value()
{
}
```

CBinarySearchTree.h

```h
#pragma once
#include "CNode.h"
#include <queue>
#include <iostream>
template<class T>
class CBinarySearchTree
{
public:
    CNode<T>* m_Root;
    CBinarySearchTree();
    ~CBinarySearchTree();
    bool Insert(T value);
    bool Remove(T value);
    void LevelOrder(CNode<T>* root);
private:
    void DeleteTree(CNode<T>* root);
    CNode<T>* GetEmptyNode(CNode<T>* root, T value);
    CNode<T>* GetDeleteNode(CNode<T>* root, T value);
    bool SetNode(CNode<T>* node, T value);
    bool DeleteNode(CNode<T>* node);
    CNode<T>* GetMaxInLeftTree(CNode<T>* root);
};
```

```cpp
template<class T>
inline CBinarySearchTree<T>::CBinarySearchTree()
{
    m_Root = new CNode<T>;
}

template<class T>
inline CBinarySearchTree<T>::~CBinarySearchTree()
{
    DeleteTree(m_Root);
}

template<class T>
inline bool CBinarySearchTree<T>::Insert(T value)
{
    CNode<T>* insertNode = GetEmptyNode(m_Root, value);
    if (!insertNode)
        return false;
    return SetNode(insertNode, value);
}
```

```cpp
template<class T>
inline bool CBinarySearchTree<T>::Remove(T value)
{
    CNode<T>* deleteNode = GetDeleteNode(m_Root, value);
    if (!deleteNode)
        return false;
    return DeleteNode(deleteNode);
}
```

```cpp
template<class T>
inline void CBinarySearchTree<T>::LevelOrder(CNode<T>* root)
{
    if (root->m_IsEmpty)
        return;
    std::queue<CNode<T>*> now;
    now.push(root);

    while (!now.empty())
    {
        CNode<T>* next = now.front();
        now.pop();
        if (next->m_IsEmpty == false) // not empty
            std::cout << next->m_Value << ", ";
        if (next->m_Left)
            now.push(next->m_Left);
        if (next->m_Right)
            now.push(next->m_Right);
    }
}
```

```cpp
template<class T>
inline void CBinarySearchTree<T>::DeleteTree(CNode<T>* root)
{
    if (root == NULL)
        return;
    DeleteTree(root->m_Left);  // 先砍左邊
    DeleteTree(root->m_Right); // 再砍右邊
    delete root; // 最後把自己砍了
}
```

```cpp
template<class T>
inline CNode<T>* CBinarySearchTree<T>::GetEmptyNode(CNode<T>* root, T value)
{
    if (root->m_IsEmpty) // 根是空的(m_IsEmpty = true)
        return root; // 直接回傳這個跟回去
    if (!root->m_IsEmpty && value == root->m_Value)
    {
        std::cout << "Replace value: " << value << std::endl; // 重複的數字出現
        return NULL;
    }
    if (value < root->m_Value)
    {
        if (root->m_Left->m_IsEmpty) // 往左子樹找
            return root->m_Left;
        else
            return GetEmptyNode(root->m_Left, value); // 繼續往左下子樹找
    }
```

```cpp
    else if (value > root->m_Value)
    {
        if (root->m_Right->m_IsEmpty) // 往右子樹找
            return root->m_Right;
        else
            return GetEmptyNode(root->m_Right, value); // 繼續往右下子樹找
    }
    return NULL; // 當作保護
}
```

```cpp
template<class T>
inline CNode<T>* CBinarySearchTree<T>::GetDeleteNode(CNode<T>* root, T value)
{
    if (root->m_IsEmpty)
        return NULL;
    if (root->m_Value == value) // 如果要刪除的數字等於根裡的數，那就回傳這個根
        return root;
    else if (value < root->m_Value) // 如果要刪除的數字小於根裡的數，那就往左走再比較
        return GetDeleteNode(root->m_Left, value);
    else if (value > root->m_Value) // 如果要刪除的數字大於根裡的數，那就往右走再比較
        return GetDeleteNode(root->m_Right, value);
    else
        return NULL; // 原則上是不會走到這裡，做一個保護
}
```

```cpp
template<class T>
inline bool CBinarySearchTree<T>::SetNode(CNode<T>* node, T value) // 設置node
{
    if(!node)
        return false;
    node->m_Left = new CNode<T>; // 先長左腳(空點)
    if (!node->m_Left)
        return false;
    node->m_Right = new CNode<T>; // 再長右腳(空點)
    if (!node->m_Right) // 如果右腳(空點)長失敗
    {
        delete node->m_Left; // 要連前面長成功的左腳(空點)也一起砍掉
        return false;
    }
    node->m_Value = value; // 放值進去囉
    node->m_IsEmpty = false; // 最後把這個node要設成非空點 (m_IsEmpty = false)
    return true;
}
```

```cpp
template<class T>
inline bool CBinarySearchTree<T>::DeleteNode(CNode<T>* node)
{
    if (node->m_Left->m_IsEmpty && node->m_Right->m_IsEmpty) // 下面沒有任何兒子了
    {
        delete node->m_Left; // 先刪左空點
        node->m_Left = NULL; // 把指標指向的地方刪掉了，要馬上把指標重新指向NULL(好習慣!)
        delete node->m_Right;// 再刪右空點
        node->m_Right = NULL;
        node->m_IsEmpty = true; // 最後把這個node要設成空點 (m_IsEmpty = true) 不管裡面有沒有值都沒差
    }
    else if (!node->m_Left->m_IsEmpty && node->m_Right->m_IsEmpty) // 有左子樹，右邊是空的(有一個兒子
    {
        T regValue = node->m_Value; // 暫存這個node裡面的數值
        node->m_Value = node->m_Left->m_Value; // 兒子取代爸爸
        if (!DeleteNode(node->m_Left)) // 如果刪除失敗
        {
            node->m_Value = regValue; // 恢復原狀
            return false; // 回傳刪除失敗
        }
    }
```

```cpp
    else if (node->m_Left->m_IsEmpty && !node->m_Right->m_IsEmpty) // 左邊是空的，有右子樹(有一個兒子
    {
        T regValue = node->m_Value; // 暫存這個node裡面的數值
        node->m_Value = node->m_Right->m_Value; // 兒子取代爸爸
        if (!DeleteNode(node->m_Right)) // 如果刪除失敗
        {
            node->m_Value = regValue; // 恢復原狀
            return false; // 回傳刪除失敗
        }
    }
    else // 有兩個兒子
    {
        CNode<T>* now = GetMaxInLeftTree(node->m_Left); // 去找左邊最大值
        T regValue = node->m_Value; // 暫存這個node裡面的數值(左邊的最大值)
        node->m_Value = now->m_Value; // 那個最大值取代爸爸
        if (!DeleteNode(now)) // 如果刪除失敗
        {
            node->m_Value = regValue; // 恢復原狀
            return false; // 回傳刪除失敗
        }
    }
```

```
    return true;
}


template<class T>
inline CNode<T>* CBinarySearchTree<T>::GetMaxInLeftTree(CNode<T>* root)
{
    CNode<T>* now = root;
    while (!now->m_Right->m_IsEmpty) // 非空點就一直往右下走
        now = now->m_Right; // 要找根的左子樹最大值，就是走到根的左子樹後，就一直往右走
    return now; // 把那個最大值的那個node傳回去
}
```

顯示結果:

```
Microsoft Visual Studio D
Insert: 41
Insert: 67
Insert: 34
Insert: 0
Insert: 69
Insert: 24
Insert: 78
Insert: 58
Insert: 62
Insert: 64
Insert: 5
Insert: 45
Insert: 81
Insert: 27
Insert: 61
Insert: 91
Insert: 95
Insert: 42
Insert: 27
Replace value: 27
Insert fail: 27
```

```
Insert: 36
Insert: 91
Replace value: 91
Insert fail: 91
Insert: 4
Insert: 2
Insert: 53
Insert: 92
Insert: 82
Insert: 21
Insert: 16
Insert: 18
Insert: 95
Replace value: 95
Insert fail: 95
Insert: 47
Insert: 26
Insert: 71
Insert: 38
Insert: 69
Replace value: 69
Insert fail: 69
Insert: 12
```

```
Insert: 67
Replace value: 67
Insert fail: 67
Insert: 99
Insert: 35
Insert: 94
Insert: 3
Insert: 11
Insert: 22
Insert: 33
Insert: 73
Insert: 64
Replace value: 64
Insert fail: 64
Insert: 41
Replace value: 41
Insert fail: 41
Insert: 11
Replace value: 11
Insert fail: 11
Insert: 53
Replace value: 53
Insert fail: 53
Insert: 68
```

```
Microsoft Visual Studio Debug Console
Insert: 53
Replace value: 53
Insert fail: 53
Insert: 68
41, 34, 67, 0, 36, 58, 69, 24, 35, 38, 45, 62, 68, 78, 5, 27, 42, 53, 61, 64, 71, 81,
 4, 21, 26, 33, 47, 73, 91, 2, 16, 22, 82, 95, 3, 12, 18, 92, 99, 11, 94,
After remove 1:
Fail to remove because there is no number 1 in the tree!
41, 34, 67, 0, 36, 58, 69, 24, 35, 38, 45, 62, 68, 78, 5, 27, 42, 53, 61, 64, 71, 81,
 4, 21, 26, 33, 47, 73, 91, 2, 16, 22, 82, 95, 3, 12, 18, 92, 99, 11, 94,
After remove 2:
Success to remove number 2 in the tree!
41, 34, 67, 0, 36, 58, 69, 24, 35, 38, 45, 62, 68, 78, 5, 27, 42, 53, 61, 64, 71, 81,
 4, 21, 26, 33, 47, 73, 91, 3, 16, 22, 82, 95, 12, 18, 92, 99, 11, 94,
After remove 16:
Success to remove number 16 in the tree!
41, 34, 67, 0, 36, 58, 69, 24, 35, 38, 45, 62, 68, 78, 5, 27, 42, 53, 61, 64, 71, 81,
 4, 21, 26, 33, 47, 73, 91, 3, 12, 22, 82, 95, 11, 18, 92, 99, 94,
```