1 测试程序的设计思路

根据提示首先编写出 detachMin 函数，大体思路与之前的 findMin 类似，通过递归找到最小节点，再将其用另一变量记录并返回后删除该节点，即返回该节点右侧的分支，若右侧无分支则返回空指针。

```
BinaryNode *detachMin(BinaryNode *&t) {
    if (t == nullptr) {
        return nullptr;
    }
    if (t->left != nullptr) {
        return detachMin(t->left);
    } else {
        BinaryNode *MinNode = t;
        t = t->right;
        return (MinNode);
    }
}
```

图 1: detachMin

接下来用指针实现 remove 函数。只需要利用上面的 detachMin 函数，在删除具有左右分支节点时搜寻其右侧分支的最小值并将其记录再删除，并且将需要删除的节点替换为该值，同时将删除节点的左右分支连接至该节点，即可实现。

```
void remove(const Comparable &x, BinaryNode *&t) {
    if (t == nullptr) {
        return;
    }
    if (x < t->element) {
        remove(x, t->left);
    } else if (x > t->element)
    {
        remove(x, t->right);
    }
    else if (t->left != nullptr && t->right != nullptr)
    {
        BinaryNode *oldNode = t;
        BinaryNode *newNode = detachMin(t->right);
        t = newNode;
        t -> right = oldNode -> right;
        t -> left = oldNode -> left;
        delete oldNode;
    } else {
        BinaryNode *oldNode = t;
        t = (t->left != nullptr) ? t->left : t->right;
        delete oldNode;
    }
}
```

图 2: remove

测试程序的设计主要分别进行了对无分支节点的删除，对有一分支节点的删除，对有两分支节点的删除以及对根节点的删除。

```cpp
void testBinarySearchTree() {
    BinarySearchTree<int> bst;
    bst.insert(5);
    bst.insert(3);
    bst.insert(7);
    std::cout << "Initial Tree:" << std::endl;
    bst.printTree();
    bst.remove(3);
    std::cout << "Tree after removing 3:" << std::endl;
    bst.printTree();
    bst.makeEmpty();
    std::cout << "Tree after making empty:" << std::endl;
    bst.printTree();

    bst.insert(10);
    bst.insert(5);
    bst.insert(15);
    bst.insert(3);
    bst.insert(7);
    bst.insert(12);
    bst.insert(18);
    std::cout << "Initial Tree:" << std::endl;
    bst.printTree();
    bst.remove(5);
    std::cout << "Tree after removing 5:" << std::endl;
    bst.printTree();
    bst.makeEmpty();
    std::cout << "Tree after making empty:" << std::endl;
    bst.printTree();

    bst.insert(21);
    bst.insert(15);
    bst.insert(35);
    bst.insert(9);
    bst.insert(28);
    bst.insert(37);
    std::cout << "Initial Tree:" << std::endl;
    bst.printTree();
    bst.remove(15);
    std::cout << "Tree after removing 15:" << std::endl;
    bst.printTree();
    bst.makeEmpty();
    std::cout << "Tree after making empty:" << std::endl;
    bst.printTree();

    bst.insert(10);
    bst.insert(5);
    bst.insert(15);
    bst.insert(3);
    bst.insert(7);
    bst.insert(12);
    bst.insert(18);
    std::cout << "Initial Tree:" << std::endl;
    bst.printTree();
    bst.remove(10);
    std::cout << "Tree after removing 10:" << std::endl;
    bst.printTree();
    bst.makeEmpty();
    std::cout << "Tree after making empty:" << std::endl;
    bst.printTree();
}
```

图 3: test_DST

2 测试的结果

测试结果一切正常。

输出为:

图 4: output

3 bug 报告

一切正常。