```
#pragma once

#include "BinarySearchTree.h"

#include <stack>

template<typename T>
class BinarySearchTreeIterator
{
private:

    using BSTree = BinarySearchTree<T>;
    using BNode = BinaryTreeNode<T>;
    using BTreeNode = BNode*;
    using BTNStack = std::stack<BTreeNode>;

    const BSTree& fBSTree;  // binary search tree
    BTNStack fStack;    // DFS traversal stack

    void pushLeft(BTreeNode aNode)
    {
        if (!aNode ->empty())
        {
            fStack.push(aNode);
            pushLeft(aNode ->left);
        }
    }

public:

    using Iterator = BinarySearchTreeIterator<T>;

    BinarySearchTreeIterator(const BSTree& aBSTree) :fBSTree(aBSTree),
      fStack()
    {
        pushLeft(aBSTree.fRoot);
    }

    const T& operator*() const
    {
        return fStack.top() ->key;
    }

    Iterator& operator++()
    {
        BTreeNode lPopped = fStack.top();
        fStack.pop();
        pushLeft(lPopped ->right);

        return *this;
    }

    Iterator operator++(int)
```

```
    {
        Iterator lTmp = *this;
        ++(*this);

        return lTmp;
    }

    bool operator==(const Iterator& aOtherIter) const
    {
        return &fBSTree == &aOtherIter.fBSTree && fStack ==
          aOtherIter.fStack;
    }

    bool operator!=(const Iterator& aOtherIter) const
    {
        return !(*this == aOtherIter);
    }

    Iterator begin() const
    {
        Iterator lTmp = *this;
        lTmp.fStack = BTNStack();
        lTmp.pushLeft(lTmp.fBSTree.fRoot);

        return lTmp;
    }

    Iterator end() const
    {
        Iterator lTmp = *this;
        lTmp.fStack = BTNStack();

        return lTmp;
    }
};
```