```
1  #pragma once
2  #include "BinarySearchTree.h"
3  #include <stack>
4  template<typename T>
5  class BinarySearchTreeIterator
6  {
7  private:
8
9      using BSTree = BinarySearchTree<T>;
10     using BNode = BinaryTreeNode<T>;
11     using BTreeNode = BNode*;
12     using BTNStack = std::stack<BTreeNode>;
13     const BSTree& fBSTree; // binary search tree
14     BTNStack fStack; // DFS traversal stack
15
16     void pushLeft(BTreeNode aNode)
17     {
18         if (!aNode->empty())
19         {
20             fStack.push(aNode);
21             pushLeft(aNode->left);
22         }
23     }
24
25 public:
26
27     using Iterator = BinarySearchTreeIterator<T>;
28
29     BinarySearchTreeIterator(const BSTree& aBSTree): fBSTree(aBSTree),
       fStack()
30     {
31         pushLeft(aBSTree.fRoot);
32     }
33     const T& operator*() const
34     {
35         return fStack.top()->key;
36     }
37     Iterator& operator++()
38     {
39         BTreeNode lPopped = fStack.top();
40         fStack.pop();
41         pushLeft(lPopped->right);
42         return *this;
43     }
44     Iterator operator++(int)
45     {
46         Iterator temp = *this;
47         ++(*this);
48         return temp;
```

```
49      }
50      bool operator==(const Iterator& aOtherIter) const
51      {
52          return &fBSTree == &aOtherIter.fBSTree && fStack ==
              aOtherIter.fStack;
53      }
54      bool operator!=(const Iterator& aOtherIter) const
55      {
56          return !(*this == aOtherIter);
57      }
58
59      Iterator begin() const
60      {
61          Iterator temp = *this;
62          temp.fStack = BTNStack();
63          temp.pushLeft(temp.fBSTree.fRoot);
64          return temp;
65      }
66      Iterator end() const
67      {
68          Iterator temp = *this;
69          temp.fStack = BTNStack();
70          return temp;
71      }
72  };
```