

```
1  #pragma once
2  #include "BinaryTreeNode.h"
3  #include <stdexcept>
4  // Problem 3 requirement
5  template<typename T>
6  class BinarySearchTreeIterator;
7  template<typename T>
8  class BinarySearchTree
9  {
10 private:
11     using BNode = BinaryTreeNode<T>;
12     using BTreeNode = BNode*;
13     BTreeNode fRoot;
14
15 public:
16     BinarySearchTree() : fRoot((&BNode::NIL)) {}
17     ~BinarySearchTree()
18     {
19         if (!fRoot->empty())
20         {
21             delete fRoot;
22         }
23     }
24     bool empty() const
25     {
26         return fRoot->empty();
27     }
28     size_t height() const
29     {
30         if (empty())
31         {
32             throw domain_error("Empty tree has no height.");
33         }
34         return fRoot->height();
35     }
36
37     bool insert(const T& aKey)
38     {
39         if (empty())
40         {
41             fRoot = new BNode(aKey);
42             return true;
43         }
44         return fRoot->insert(aKey);
45     }
46     bool remove(const T& aKey)
47     {
48         if (empty())
49         {
```

```
50         throw domain_error("Cannot remove in empty tree.");
51     }
52     if (fRoot->leaf())
53     {
54         if (fRoot->key != aKey)
55         {
56             return false;
57         }
58         fRoot = &BNode::NIL;
59         return true;
60     }
61     return fRoot->remove(aKey, &BNode::NIL);
62 }
63 // Problem 3 methods
64
65 using Iterator = BinarySearchTreeIterator<T>;
66 // Allow iterator to access private member variables
67 friend class BinarySearchTreeIterator<T>;
68 Iterator begin() const
69 {
70     return Iterator(*this).begin();
71 }
72 Iterator end() const
73 {
74     return Iterator(*this).end();
75 }
76 };
```