

```
1
2 // COS30008, Final Exam
3
4 #pragma once
5
6 #include "TernaryTree.h"
7
8 #include <stack>
9
10 template<typename T>
11 class TernaryTreePrefixIterator
12 {
13 private:
14     using TTree = TernaryTree<T>;
15     using TTreeNode = TTree*;
16     using TTreeStack = std::stack<const TTree*>;
17
18     const TTree* fTTree;           // ternary tree
19     TTreeStack fStack;             // traversal stack
20
21 public:
22
23     using Iterator = TernaryTreePrefixIterator<T>;
24
25     Iterator operator++(int)
26     {
27         Iterator old = *this;
28
29         ++(*this);
30
31         return old;
32     }
33
34     bool operator!=(const Iterator& aOtherIter) const
35     {
36         return !(*this == aOtherIter);
37     }
38
39     //////////////////////////////////////
40     // Problem 4: TernaryTree Prefix Iterator
41
42 private:
43
44     // push subtree of aNode [30]
45     void push_subtrees(const TTree* aNode)
46     {
47         if (!aNode->getRight().empty())
48         {
49             fStack.push(&aNode->getRight());
```

```
50     }
51     if (!aNode->getMiddle().empty())
52     {
53         fStack.push(&aNode->getMiddle());
54     }
55     if (!aNode->getLeft().empty())
56     {
57         fStack.push(&aNode->getLeft());
58     }
59 }
60
61 public:
62
63     // iterator constructor [12]
64     TernaryTreePrefixIterator( const TTree* aTTree ) :
65         fTTree(aTTree)
66     {
67         if (fTTree != &TTree::NIL)
68         {
69             fStack.push(fTTree);
70         }
71     }
72
73     // iterator dereference [8]
74     const T& operator*() const
75     {
76         return **fStack.top();
77     }
78
79     // prefix increment [12]
80     Iterator& operator++()
81     {
82         if (!fStack.empty())
83         {
84             const TTree* current = fStack.top();
85             fStack.pop();
86             push_subtrees(current);
87         }
88
89         return *this;
90     }
91
92     // iterator equivalence [12]
93     bool operator==(const Iterator& aOtherIter) const
94     {
95         return (fTTree == aOtherIter.fTTree)
96             && (fStack == aOtherIter.fStack);
97     }
98
```

```
99     // auxiliaries [4,10]
100     Iterator begin() const
101     {
102         return Iterator(fTTree);
103     }
104
105     Iterator end() const
106     {
107         Iterator lIter(fTTree);
108         lIter.fStack = TTreeStack();
109         return lIter;
110     }
111 };
112
```