```
 1
 2  // COS30008, Final Exam, 2022
 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(const_cast<TTreeNode>(&(*aNode).getRight()));
```

```
50          }
51          if (!(*aNode).getMiddle().empty())
52          {
53              fStack.push(const_cast<TTreeNode>(&(*aNode).getMiddle()));
54          }
55          if (!(*aNode).getLeft().empty())
56          {
57              fStack.push(const_cast<TTreeNode>(&(*aNode).getLeft())); 5;
58          }
59      }
60
61  public:
62
63      // iterator constructor [12]
64      TernaryTreePrefixIterator( const TTree* aTTree ): fTTree(aTTree),
          fStack()
65      {
66          if (!(*fTTree).empty())
67          {
68              fStack.push(const_cast<TTreeNode>(fTTree));
69          }
70      }
71
72      // iterator dereference [8]
73      const T& operator*() const
74      {
75          return **fStack.top();
76      }
77
78      // prefix increment [12]
79      Iterator& operator++()
80      {
81          TTreeNode lPopped = const_cast<TTreeNode>(fStack.top());
82          fStack.pop();
83          push_subtrees(lPopped);
84          return *this;
85      }
86
87      // iterator equivalence [12]
88      bool operator==( const Iterator& aOtherIter ) const
89      {
90          return fTTree == aOtherIter.fTTree && fStack.size() ==
              aOtherIter.fStack.size();
91      }
92
93      // auxiliaries [4,10]
94      Iterator begin() const
95      {
96          Iterator temp = *this;
```

```
 97          temp.fStack = TTreeStack();
 98          temp.fStack.push(const_cast<TTreeNode>(temp.fTTree));
 99          return temp;
100      }
101      Iterator end() const
102      {
103          Iterator temp = *this;
104          temp.fStack = TTreeStack();
105          return temp;
106      }
107  };
108
```