```
 1  #pragma once
 2  #include <stdexcept>
 3  #include <algorithm>
 4  using namespace std;
 5  template<typename T>
 6  struct BinaryTreeNode
 7  {
 8      using BNode = BinaryTreeNode<T>;
 9      using BTreeNode = BNode*;
10
11      T key;
12      BTreeNode left;
13      BTreeNode right;
14
15      static BNode NIL;
16      const T& findMax() const
17      {
18          if (empty())
19          {
20              throw domain_error("Empty tree encountered");
21          }
22          if (right->empty())
23          {
24              return key;
25          }
26          return right->findMax();
27      }
28      const T& findMin() const
29      {
30          if (empty())
31          {
32              throw domain_error("Empty tree encountered");
33          }
34          if (left->empty())
35          {
36              return key;
37          }
38          return left->findMin();
39      }
40      bool remove(const T& aKey, BTreeNode aParent)
41      {
42          BTreeNode x = this;
43          BTreeNode y = aParent;
44          while (!x->empty())
45          {
46              if (aKey == x->key)
47              {
48                  break;
49              }
```

```
50              y = x;
51              x = aKey < x->key ? x->left : x->right;
52          }
53          if (x->empty())
54          {
55              return false;
56          }
57
58          if (!x->left->empty())
59          {
60              const T& lKey = x->left->findMax();
61              x->key = lKey;
62              x->left->remove(lKey, x);
63          }
64          else
65          {
66              if (!x->right->empty())
67              {
68                  const T& lKey = x->right->findMin();
69                  x->key = lKey;
70                  x->right->remove(lKey, x);
71              }
72              else
73              {
74                  if (y != &NIL)
75                  {
76                      if (y->left == x)
77                      {
78                          y->left = &NIL;
79                      }
80                      else
81                      {
82                          y->right = &NIL;
83                      }
84                  }
85                  delete x;
86              }
87          }
88          return true;
89      }
90
91      BinaryTreeNode(): key(T()), left(&NIL), right(&NIL){}
92      BinaryTreeNode(const T& aKey): key(aKey), left(&NIL), right(&NIL){}
93      BinaryTreeNode(T&& aKey): key(move(aKey)), left(&NIL), right(&NIL){}
94      ~BinaryTreeNode()
95      {
96          if (!left->empty())
97          {
98              delete left;
```

```
 99          }
100          if (!right->empty())
101          {
102              delete right;
103          }
104      }
105
106      bool empty() const
107      {
108          return this == &NIL;
109      }
110      bool leaf() const
111      {
112          return left->empty() && right->empty();
113      }
114      size_t height() const
115      {
116          if (empty())
117          {
118              throw domain_error("Empty tree encountered");
119          }
120          if (leaf())
121          {
122              return 0;
123          }
124          const int left_height = left->empty() ? 1 : left->height() + 1;
125          const int right_height = right->empty() ? 1 : right->height() + 1;
126          return max(left_height, right_height);
127      }
128      bool insert(const T& aKey)
129      {
130          if (empty())
131          {
132              return false;
133          }
134          if (aKey > key)
135          {
136              if (right->empty())
137              {
138                  right = new BNode(aKey);
139              }
140              else
141              {
142                  return right->insert(aKey);
143              }
144              return true;
145          }
146          if (aKey < key)
147          {
```

```
148             if (left->empty())
149             {
150                 left = new BNode(aKey);
151             }
152             else
153             {
154                 return left->insert(aKey);
155             }
156             return true;
157         }
158         return false;
159     }
160 };
161 template<typename T>
162 BinaryTreeNode<T> BinaryTreeNode<T>::NIL;
```