```cpp
#include "btNode.h"

void bst_insert(btNode*& bst_root, int insInt)
{
    if (bst_root == 0) // check if root is null
    {
        bst_root = new btNode;
        bst_root->data = insInt;
        bst_root->left = 0;
        bst_root->right = 0;

        return;
    }

    btNode* temp = bst_root;

    while (temp != 0)
    {
        if (temp->data == insInt) // insInt match
        {
            return;
        }

        if (temp->data < insInt) // insInt goes on right branch
        {
            if (temp->right == 0) // vacant branch on right
            {
                temp->right = new btNode;
                temp->right->data = insInt;
                temp->right->left = 0;
                temp->right->right = 0;
            }
            else
            {
                temp = temp->right;
            }

            continue;
        }

        // insInt goes on left branch
        if (temp->left == 0) //vacant branch on left
        {
            temp->left = new btNode;
            temp->left->data = insInt;
            temp->left->left = 0;
            temp->left->right = 0;
        }
        else
```

```
50          {
51              temp = temp->left;
52          }
53      }
54  }
55
56  bool bst_remove(btNode*& bst_root, int delInt)
57  {
58      if (bst_root == 0) // check if tree is empty
59      {
60          return false;
61      }
62
63      if (bst_root->data > delInt) // recur left
64      {
65          return bst_remove(bst_root->left, delInt);
66      }
67
68      if (bst_root->data < delInt) //recur right
69      {
70          return bst_remove(bst_root->right, delInt);
71      }
72
73      if (bst_root->data == delInt)
74      {
75          btNode* old_bst_root = bst_root;
76
77          if (bst_root->left != 0 && bst_root != 0) // two children
78          {
79              bst_remove_max(bst_root->left, bst_root->data);
80              return true;
81          }
82
83          if (bst_root->left == 0 && bst_root->right != 0) // no left child
84          {
85              bst_root = bst_root->right;
86              delete old_bst_root;
87          }
88          else if (bst_root->left != 0 && bst_root->right == 0) // no right
              child
89          {
90              bst_root = bst_root->left;
91              delete old_bst_root;
92          }
93          else // no children
94          {
95              bst_root = 0;
96          }
97
```

```
 98          return true;
 99      }
100  }
101
102  void bst_remove_max(btNode*& bst_root, int& data)
103  {
104      if (bst_root == 0) // check if tree is empty
105      {
106          return;
107      }
108
109      if (bst_root->right == 0) // check if root has right child
110      {
111          data = bst_root->data;
112
113          btNode* temp = bst_root;
114          bst_root = bst_root->left;
115          delete temp;
116
117          return;
118      }
119
120      bst_remove_max(bst_root->right, data);
121  }
122
123
124  void portToArrayInOrder(btNode* bst_root, int* portArray)
125  {
126     if (bst_root == 0) return;
127     int portIndex = 0;
128     portToArrayInOrderAux(bst_root, portArray, portIndex);
129  }
130
131  void portToArrayInOrderAux(btNode* bst_root, int* portArray, int&
       portIndex)
132  {
133     if (bst_root == 0) return;
134     portToArrayInOrderAux(bst_root->left, portArray, portIndex);
135     portArray[portIndex++] = bst_root->data;
136     portToArrayInOrderAux(bst_root->right, portArray, portIndex);
137  }
138
139  void tree_clear(btNode*& root)
140  {
141     if (root == 0) return;
142     tree_clear(root->left);
143     tree_clear(root->right);
144     delete root;
145     root = 0;
```

```
146 }
147
148 int bst_size(btNode* bst_root)
149 {
150    if (bst_root == 0) return 0;
151    return 1 + bst_size(bst_root->left) + bst_size(bst_root->right);
152 }
153
```