

```

1  #ifndef BT_NODE_H
2  #define BT_NODE_H
3
4  struct btNode
5  {
6      int data;
7      btNode* left;
8      btNode* right;
9  };
10
11 // pre:  bst_root is root pointer of a binary search tree (may be 0 for
12 //        empty tree) and dumpArray has the base address of an array large
13 //        enough to hold all the data items in the binary search tree
14 // post:  The binary search tree has been traversed in-order and the data
15 //        values are written (as they are encountered) to dumpArray in
16 //        increasing positional order starting from the first element
17 void dumpToArrayInOrder(btNode* bst_root, int* dumpArray);
18 void dumpToArrayInOrderAux(btNode* bst_root, int* dumpArray, int& dumpIndex);
19
20 // pre:  (none)
21 // post:  dynamic memory of all the nodes of the tree rooted at root has been
22 //        freed up (returned back to heap/freestore) and the tree is now empty
23 //        (root pointer contains the null address)
24 void tree_clear(btNode*& root);
25
26 // pre:  (none)
27 // post:  # of nodes contained in tree rooted at root is returned
28 int bst_size(btNode* bst_root);
29
30 ///////////////////////////////////////////////////
31
32 // pre:  bst_root is root pointer of a binary search tree (may be 0 for
33 //        empty tree)
34 // post:  If no node in the binary search tree has data equals insInt, a
35 //        node with data insInt has been created and inserted at the proper
36 //        location in the tree to maintain binary search tree property.
37 //        If a node with data equals insInt is found, the node's data field
38 //        has been overwritten with insInt; no new node has been created.
39 // write prototype for bst_insert here
40 void bst_insert(btNode*& bst_root, int insInt);
41
42 // pre:  bst_root is root pointer of a binary search tree (may be 0 for
43 //        empty tree)
44 // post:  If remInt was in the tree, then remInt has been removed, bst_root
45 //        now points to the root of the new (smaller) binary search tree,
46 //        and the function returns true. Otherwise, if remInt was not in the
47 //        tree, then the tree is unchanged, and the function returns false.
48 // write prototype for bst_remove here
49 bool bst_remove(btNode*& bst_root, int remInt);
50
51 // pre:  bst_root is root pointer of a non-empty binary search tree
52 // post:  The largest item in the binary search tree has been removed, and
53 //        bst_root now points to the root of the new (smaller) binary search
54 //        tree. The reference parameter, removed, has been set to a copy of
55 //        the removed item.
56 // write prototype for bst_remove_max here
57 void bst_remove_max(btNode*& bst_root, int& item);
58
59 #endif
60

```