#include "btNode.h"

void dumpToArrayInOrder(btNode\* bst\_root, int\* dumpArray)

{

if (bst\_root == 0) return;

int dumpIndex = 0;

dumpToArrayInOrderAux(bst\_root, dumpArray, dumpIndex);

}

void dumpToArrayInOrderAux(btNode\* bst\_root, int\* dumpArray, int& dumpIndex)

{

if (bst\_root == 0) return;

dumpToArrayInOrderAux(bst\_root->left, dumpArray, dumpIndex);

dumpArray[dumpIndex++] = bst\_root->data;

dumpToArrayInOrderAux(bst\_root->right, dumpArray, dumpIndex);

}

void tree\_clear(btNode\*& root)

{

if (root == 0) return;

tree\_clear(root->left);

tree\_clear(root->right);

delete root;

root = 0;

}

int bst\_size(btNode\* bst\_root)

{

if (bst\_root == 0) return 0;

return 1 + bst\_size(bst\_root->left) + bst\_size(bst\_root->right);

}

void bst\_insert(btNode\*& bst\_root, int insInt)

{

if(bst\_root == nullptr)

{

btNode\* new\_root = new btNode;

new\_root->data = insInt;

new\_root->left = new\_root->right = nullptr;

bst\_root = new\_root;

return;

}

btNode\* nodePtr = bst\_root;

while (nodePtr != nullptr)

{

if (nodePtr->data > insInt)

{

if(nodePtr->left == nullptr)

{

nodePtr->left = new btNode;

nodePtr->left->data = insInt;

nodePtr->left->left = nodePtr->left->right = nullptr;

return;

}

else

nodePtr = nodePtr->left;

}

else if(nodePtr->data < insInt)

{

if(nodePtr->right == nullptr)

{

nodePtr->right = new btNode;

nodePtr->right->data = insInt;

nodePtr->right->left = nodePtr->right->right = nullptr;

return;

}

else

nodePtr = nodePtr->right;

}

else

return;

}

}

bool bst\_remove(btNode\*& bst\_root, int remInt)

{

if(bst\_root == nullptr)

return false;

if(bst\_root->data > remInt)

{

bst\_remove(bst\_root->left, remInt);

}

else if(bst\_root->data < remInt)

{

bst\_remove(bst\_root->right, remInt);

}

else

{

if(bst\_root->left != nullptr && bst\_root->right != nullptr)

{

bst\_remove\_max(bst\_root->left, bst\_root->data);

}

else

{

btNode\* old\_bst\_root = bst\_root;

if(bst\_root->left == nullptr && bst\_root->right != nullptr)

{

bst\_root = bst\_root->right;

}

else if (bst\_root-> left != nullptr && bst\_root->right == nullptr)

{

bst\_root = bst\_root->left;

}

else

bst\_root = nullptr;

delete old\_bst\_root;

}

return true;

}

}

void bst\_remove\_max(btNode\*& bst\_root, int& data)

{

if(bst\_root == nullptr)

return;

if(bst\_root->right == nullptr)

{

btNode\* deleteNode = bst\_root;

data = bst\_root->data;

bst\_root = bst\_root->left;

delete deleteNode;

}

else

bst\_remove\_max(bst\_root->right, data);

}