template <class ItemType>

void DeleteNode (TreeNode <ItemType> \* & tree, bool & shorter)

// Delete the node pointed to by tree.

// Post: The user's data in the node pointed to by tree is no

// longer in the tree. If tree is a leaf node or has only

// one non-NULL child pointer, the node pointed to by tree is

// deleted; otherwise, the user's data is replaced by its

// logical predecessor and the predecessor's node is deleted.

{

ItemType data;

TreeNode <ItemType> \* tempPtr;

tempPtr = tree;

if (tree->left == NULL)

{

tree = tree->right;

delete tempPtr;

shorter = true;

}

else if (tree->right == NULL)

{

tree = tree->left;

delete tempPtr;

shorter = true;

}

else

{

GetPredecessor(tree, data);

tree->info = data;

Delete (tree->left, data, shorter); // Delete the predecessor node

if (shorter)

switch (tree->bf)

{

case LH: tree->bf = EH; break;

case EH: tree->bf = RH; shorter = false; break;

case RH: DelRightBalance(tree, shorter);

}

}

}

template <class ItemType>

void DelRightBalance (TreeNode<ItemType> \*& tree, bool & shorter)

{

TreeNode<ItemType> \* rs = tree->right;

TreeNode<ItemType> \* ls;

switch (rs->bf)

{

case RH: tree->bf = rs->bf = EH;

RotateLeft(tree);

shorter = true;

break;

case EH: tree->bf = RH;

rs->bf = LH;

RotateLeft(tree);

shorter = false;

break;

case LH: ls = rs->left;

switch (ls->bf)

{

case RH: tree->bf = LH;

rs->bf = EH;

break;

case EH: tree->bf = rs->bf = EH;

break;

case LH: tree->bf = EH;

rs->bf = RH;

break;

}

ls->bf = EH;

RotateRight(tree->right);

RotateLeft(tree);

shorter = true;

}

}

template <class ItemType>

void DelLeftBalance (TreeNode<ItemType> \*& tree, bool & shorter)

{

TreeNode<ItemType> \* ls = tree->left;

TreeNode<ItemType> \* rs;

switch (ls->bf)

{

case LH: tree->bf = ls->bf = EH;

RotateRight(tree);

shorter = true;

break;

case EH: tree->bf = LH;

ls->bf = RH;

RotateRight(tree);

shorter = false;

break;

case RH: rs = ls->left;

switch (rs->bf)

{

case LH: tree->bf = RH;

ls->bf = EH;

break;

case EH: tree->bf = ls->bf = EH;

break;

case RH: tree->bf = EH;

ls->bf = LH;

break;

}

rs->bf = EH;

RotateLeft(tree->left);

RotateRight(tree);

shorter = true;

}

}