typedef struct {
    int data;
    node *left;
    node *right;
} node;

// Deletes value from a BST rooted at root. value must be in the tree in
// to work. Returns a pointer to the root of the resulting tree.
node* delete(node* root, int value) {
    node *delnode, *new_del_node, *save_node, *par;
    int save_val;

    delnode = _____________________; // Get a pointer to the node to delete.
    par = __________________________; // Get the parent of this node.

    // Case 1: the node to delete is a leaf node.
    if (____________________) {

        // Deleting the only node in the tree.
        if (par == NULL) {
            _________________; // free the memory for the node.
            return ________________;
        }

        // Deletes the node if it's a left child.
        if (____________________) {
            free(_______________); // Free the memory for the node.
            _________________ = NULL;
        }

        // Deletes the node if it's a right child.
        else {
            free(_______________); // Free the memory for the node.
            _________________ = NULL;
        }

        return root; // Return the root of the new tree.
    }

    // Case 2: the node to be deleted only has a left child.
    if (____________________________) {

        // Deleting the root node of the tree.
        if (____________________) {
            save_node = _________________;
            free(_______________); // Free the node to delete.
            return ________________; // Return the new root node of the resulting tree.
        }

        // Deletes the node if it's a left child.
if (________________________) {
    save_node = _________________; // Save the node to delete.
    par->left = _________________; // Readjust the parent pointer.
    free(________________________); // Free the memory for the deleted node.
}
// Deletes the node if it's a right child.
else {
    save_node = _________________; // Save the node to delete.
    par->right = _________________; // Readjust the parent pointer.
    free(________________________); // Free the memory for the deleted node.
}

return root; // Return the root of the tree after the deletion.
}

// Case 3: the node to be deleted only has a right child.
if (________________________) {

    // Node to delete is the root node.
    if (par == NULL) {
        save_node = _________________;
        free(________________________);
        return ______________________;
    }

    // Delete's the node if it is a left child.
    if (________________________) {
        save_node = _________________;
        par->left = _________________;
        free(________________________);
    }

    // Delete's the node if it is a right child.
    else {
        save_node = _________________;
        par->right = _________________;
        free(________________________);
    }
    return root;
}

// Case 4: The deleted node has 2 children, find the replacement node
new_del_node = ______________________;
save_val = ______________________;
delete(root, _________________); // Now, delete the proper value.

// Restore the data to the original node to be deleted.
delnodel->data = _________________;

return root;
}