15-122: Principles of Imperative Computation

Recitation 16 Solutions — Josh Zimmerman

Proof that `tree_insert`'s postcondition holds.

```
 1 tree* tree_insert(tree* T, elem e)
 2 //@requires is_ordtree(T);
 3 //@requires e != NULL;
 4 //@ensures is_ordtree(\result);
 5 {
 6    if (T == NULL) {
 7       /* create new node and return it */
 8       T = alloc(struct tree_node);
 9       T->data = e;
10       T->left = NULL;
11       T->right = NULL;
12       return T;
13    }
14    int r = key_compare(elem_key(e), elem_key(T->data));
15    if (r == 0) {
16       T->data = e;   /* modify in place */
17    }
18    else if (r < 0) {
19       T->left = tree_insert(T->left, e);
20    }
21    else {
22       //@assert r > 0;
23       T->right = tree_insert(T->right, e);
24    }
25    return T;
26 }
```

To help you understand how `tree_insert` works, write a proof that it works.

As a reminder, when showing that recursive functions are correct, we first show partial correctness by first showing that in the base case the function is correct and then showing that if the function is correct on a smaller case it is correct on a larger case. We then show termination.

In this case, our steps will be as follows:

1. Show that the base case of the code (the case when `T == NULL`) is correct.
2. Show that if the recursive calls we make in the function (lines 19 and 23) are correct, then the function as a whole is correct.
3. Show that the function terminates on all input.

Solution:

Base case. When `T == NULL`, we return a tree that only has one element. This element satisfies the basic data structure invariants, and is correctly ordered since there is only one node.

Inductive hypothesis. `tree_insert(T->left, e)` and `tree_insert(T->right, e)` satisfy their post-condition, for some tree `T`.

Inductive step. We have three cases. `r == 0`, `r < 0`, or `r > 0`.

In the first case, we don't modify any keys in the tree so the order invariant is preserved. Also, we know `e` is not `NULL`, so the other invariants are correct as well.

In the second case, we know that the key we wish to insert is smaller than the key of the root of the sub-tree we're considering. So, the element we wish to insert should go to the left of this tree.

By the inductive hypothesis, `tree_insert(T->left, e)` will return a valid `ordtree`. Since `e` should go on the left side of the tree, and since by the precondition `is_ordtree(T)`, we know that everything on the left of the tree is smaller than the current root node.

In the third case, we know that the key we wish to insert is larger than the key of the root of the sub-tree we're considering. So, we know that `e` should go on the right side.

By our inductive hypothesis, `tree_insert(T->right, e)` satisfies its postcondition. Since we know everything in the right subtree is larger than the current root node, the invariants still hold.

Termination. At every stage, we recurse with a subtree of the tree. Assuming that there are no cycles in the tree, we'll eventually reach a node that has no children, and thus will reach a `NULL` node, at which point we terminate. (Note that if we have a cycle in the BST, this doesn't terminate, so it's important that we don't have any cycles.)