1) (10 pts) DSN (Binary Trees)

Demonstrate your understanding of recursion by rewriting the following recursive function, which operates on a binary tree, as an **iterative** function. In doing so, you must abide by the following restrictions:

1. Do not use *break* statements in your solution.
2. Do not write any helper functions in your solution.
3. Do not make any recursive calls in your solution.

```
int foo(node *root) {
    if (root == NULL) return 1;
    if (root->left == NULL && root->right == NULL) return 2;
    if (root->left == NULL) return 3 * foo(root->right);
    if (root->right == NULL) return 4 * foo(root->left);
    if (root->right->data > root->left->data) return 5 * foo(root->right);
    return 6 * foo(root->left);
}

int iterative_foo(node *root) {

      int result = 1;
      node *temp = root;

      while (temp != NULL) {
            if (temp->left == NULL && temp->right == NULL) {
               result *= 2;
               temp = NULL;
            }
            else if (temp->left == NULL) {
               result *= 3;
               temp = temp->right;
            }
            else if (temp->right == NULL) {
               result *= 4;
               temp = temp->left;
            }
            else if (temp->right->data > temp->left->data) {
               result *= 5;
               temp = temp->right;
            }
            else {
               result *= 6;
               temp = temp->left;
            }
      }

      return result;
}
```

See grading notes on following page.

Grading for Question #1:

Note: The *temp* variable is not strictly necessary. We could instead use *root* to traverse the tree. Since it's just a local copy of the original *root* pointer at the start of the function, modifying it has no adverse impact here.

Note: We cannot check whether *temp == NULL* in the while loop, since it's gated by the condition that *temp != NULL*.

+1 pt for creating a result variable of some sort.
+1 pt for initializing the result variable to 1. (Initializing to 0 would wipe out results of multiplication. If they found a way around that, though, award this point.)
+1 pt for looping condition. They might instead use *while(1)* and then check whether *temp == NULL* inside the loop, returning *result* if so. That alternative approach is fine. If they forget to declare temp or initialize *temp* to *root*, or if the data type is wrong for that variable, do not award this 1 pt.
+2 pts for moving pointer left and right as appropriate in all cases.
+2 pts for multiplying result variable by appropriate integer in all cases.
+2 for overall structure and general correctness of solution (i.e., a while loop with if statements that multiply the return value by some integer and move a pointer left or right). Note that if they use a temp variable, they must be careful to actually use that in their *if* conditions (rather than *root*), as well as their assignment statements (e.g., temp = temp->right, not temp = root->right). Note also that they **must** use *else* statements.to prevent multiple *if* statements from triggering. The recursive version uses *return* statements to prevent that, but in an iterative version, we can't just slap an *else* statement on the last condition. Otherwise, we could check multiple conditions and possibly encounter a segmentation fault.
+1 for finding a way to terminate when root->left and root->right are both NULL. This could involve a return statement within the loop, or they could set their pointer to NULL.