Checkpoint 0

Write a function to reverse a queue, using only the functions from the stack and queue interfaces.

```c
void reverse(queue_t Q) {
    stack_t S = stack_new(); // Hint: Allocate a temporary data structure
    while (!queue_empty(Q)) {
        string temp = deq(Q);
        push(S, temp);
    }
    while (!stack_empty(S)) {
        string temp = pop(S);
        enq(Q, temp);
    }
}
```

Checkpoint 1

Write a recursive function to count the size of a stack.

```c
int size(stack_t S) {
    if (stack_empty(S)) return 0; 
    string x = pop(S); 
    int i = size(S); 
    i++; 
    push(S, x); 
    return i; 
}
```

Checkpoint 2

Why couldn’t this stack size implementation be used in contracts in C0?
Solution: C0 has checks in place to make sure that any functions that are called from the assertion language (//@requires, //@ensures, //@loop_invariant, or //@assert) are pure – that is, they don’t manipulate memory. This is to ensure that code running with contracts will always return the same answer as code running without contracts. (It’s possible to turn off this check with the –no-purity-check option.)

Because pushing and popping from the stack modifies allocated memory (we haven’t seen how, yet, but we know, even with our abstract picture of stacks, that it must), the size function we wrote above isn’t pure.

Checkpoint 3

The above example works because function calls use a data structure that is like a stack. Step by step, trace out operationally the state of the computer’s memory when it calculates the size of a stack with two strings “b” and “c”, taking account of the fact that each recursive call gets its own copy of the assignable variables.

Solution:
Checkpoint 4

In the same fashion, trace out what happens operationally in this broken reversal function, starting with the code in main().

1
 void reverse(stack_t S) {
2    string x;
```c
stack_t R = stack_new();
while (!stack_empty(S)) {
    x = pop(S);
    push(R, x);
}

S = R;
}

int main() {
    stack_t S = stack_new();
    push(S, "foo");
    reverse(S);
    println(pop(S));
    return 0;
}
```

*** Safety violation! The stack S is empty.***