**Recursion Aversion**

I’ve thought of a great way to manage my aversion to recursion. I’m going to write a simple and cute NON-recursive program to generate the insomniac bunny story—you know, “Once upon a time there was a Baby bunny who just couldn’t fall asleep, so Mommy rabbit read the Baby bunny a story about a Baby bear who just couldn’t fall asleep, so Mommy bear read the Baby bear a story about …,” and so on until you finally encounter a story in which a baby animal finally falls fast asleep, and sure enough the stack unwinds, and sure enough the Mommy rabbit finally gets some sleep. THEN I will write a recursive program about a programmer who just couldn’t bring himself to deal with recursion, so he wrote a program about a programmer who—Oh, never mind, I need to get some sleep first.

It’s been a long and lovely day. **SO ANYWAY…**

Once upon a time there was a baby rabbit who just couldn't fall asleep, so the mommy rabbit read little baby rabbit a story about a baby bear who just couldn't fall asleep, so the mommy bear read little baby bear a story about a baby spider who just couldn't fall asleep, so the mommy spider read little baby spider a story about a baby shark who just couldn't fall asleep, so the mommy shark read little baby shark a story about a baby lion who just couldn't fall asleep, and sure enough the baby lion fell fast asleep, and the baby shark fell fast asleep, and the baby spider fell fast asleep, and the baby bear fell fast asleep, and mommy rabbit finally got some sleep.

Once upon a time there was a baby rabbit who just couldn't fall asleep, so the mommy rabbit read little baby rabbit a story about a baby bear who just couldn't fall asleep, so the mommy bear read little baby bear a story about a baby spider who just couldn't fall asleep, so the mommy spider read little baby spider a story about a baby shark who just couldn't fall asleep, so the mommy shark read little baby shark a story about a baby lion who just couldn't fall asleep, and sure enough, the baby lion fell fast asleep, and the baby shark fell fast asleep, and the baby spider fell fast asleep, and the baby bear fell fast asleep, and mommy rabbit finally got some sleep.

Once upon a time there was a baby rabbit who just couldn't fall asleep, so the mommy rabbit read little baby rabbit a story about a baby bear who just couldn't fall asleep, so the mommy bear read little baby bear a story about a baby spider who just couldn't fall asleep, so the mommy spider read little baby spider a story about a baby shark who just couldn't fall asleep, so the mommy shark read little baby shark a story about a baby lion who just couldn't fall asleep, and sure enough, the baby lion fell fast asleep, and the baby shark fell fast asleep, and the baby spider fell fast asleep, and the baby bear fell fast asleep, and mommy rabbit finally got some sleep.

Once upon a time there was a baby rabbit who just couldn't fall asleep, so the mommy rabbit read little baby rabbit a story about a baby bear who just couldn't fall asleep, so the mommy bear read little baby bear a story about a baby spider who just couldn't fall asleep, so the mommy spider read little baby spider a story about a baby shark who just couldn't fall asleep, so the mommy shark read little baby shark a story about a baby lion who loved to fall asleep listening to a lovely bedtime story, and sure enough, the baby lion fell fast asleep, and the baby shark fell fast asleep, and the baby spider fell fast asleep, and the baby bear fell fast asleep, and mommy rabbit finally got some sleep. 1/14/2018 5:41:02.

**ANYWAY AND FURTHERMORE, even broken code has its instructional charms, so read on:**

**Recursion Therapy**

I kept at writing a sleepy-bear program, and I feel a little better now. Here’s what I learned. VBA can only recurse to a maximum depth of about 6,000 or 7,000 calls. Tail recursion and head recursion both max out at the same depth. No optimization for tail recursion takes place.

Furthermore, I learned that the sleepy-bear program is a great example of head recursion COMBINED WITH tail recursion. Three phases of recursion take place: first, you deepen the story; second, you report the bottom-level of the story, third you wrap up each previous level of the story. Phase One performs head recursion to push each insomniac animal onto the stack. Phase Two occurs when the stack hits bottom—recursion reaches its pre-determined maximum depth and reports a sleepy animal who buys the story. Phase Three performs tail recursion to unwind the stack; each insomniac sub-story gets a happy ending all the way back up the stack—until the original baby bear finally falls asleep.

I built and tested the sleepy-bear idea in stages. First, I made a NON-recursive program consisting of 2 separate, consecutive FOR-NEXT loops that performed the string-handling. My first loop introduced insomniac animals 1 through 4; then I put a 5th animal to sleep between the two loops; then I looped out, putting animals 4 through 1 to sleep. In the second stage, I wrote a heads-and-tails numeric recursion framework that merely counted from 1 to 20 going in and counted back from 20 to 1 coming out:



Third, I fleshed out this numeric recursion framework to concatenate boiler-plate narrative text using reasonably cool animal names taken from a hard-coded string array:



String concatenation looks naturally stinky, so I recommend reading the simpler numeric recursion subroutine first, to visualize what heads-and-tails recursion actually does. Note that my simple numeric recursive subroutine outputs the final stack value, 20, twice—which does not serve the animal narrative correctly. It’s a worthwhile exercise to fix this numeric recursion framework so that it puts the number 20 to sleep only once!

Next consider my claim that VBA recursion maxes out with a stack-overflow error after 6,000 or 7,000 calls. I verified this claim by stress-testing my numeric framework subroutine whilst pushing single integers onto the stack during each recursion. What if I had decided to push the entire “story-so-far” onto the sleepy animal stack? (In reality, I choose to make my text-narrative recursion perform its cumulative work upon a general declaration variable, BedtimeStory, outside the recursion stack.) For sure, a fat string stack would never make it to 6,000 calls! On the other hand, the fat stack would provide its own opportunity to observe and experiment with how much raw stack space VBA is actually

willing to allocate. (C# has its own allocation attitudes, but I digress. Don’t even get me started on the charms of the .NET StringBuilder object)

In summary, I think creating a deliberately string-clogged VBA stack is a worthwhile exercise—up to a point. Deliberate stack-clogging is a way to make recursion feel physical, not just logical.

After all this, do I like recursion better now than I used to? Well yes, but only a little.



**© 2018 PraxisMachineCodingLessons.com**