Fun with Languages — Jason Nguyen (XXXXXXX)

It's been quite the wild ride being in this course, having learned COBOL, Ada, and Fortran. I chose task three, the Russian Peasant Multiplication implementation. I ended up having a lot of fun with this and ended up spending way too much time messing around with different implementations. I'll get to that later. This is a table showing the different runtimes of the programs in seconds.

Test	1 to 5	5 to 50	50 to 500	500 to 5000
C, Recursive	0.000008	0.000184	0.017642	1.269799
C, Iterative	0.000004	0.000136	0.006498	0.793342
Ada, Recursive	0.000017	0.000187	0.018023	1.2385
Ada, Iterative	0.000006	0.000162	0.007478	0.88529
Fortran, Recursive	0.000003	0.000198	0.02247	1.431143
Fortran, Iterative	0.000003	0.000157	0.010114	0.856655

Here are my individual experiences:

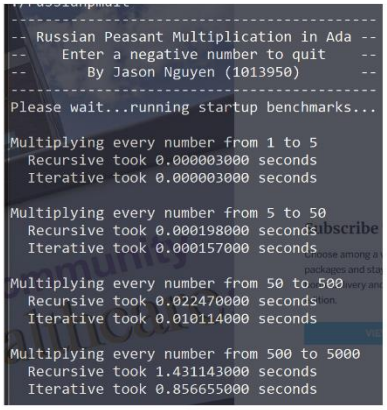
- **C**—the easiest one by far. My favourite language, I spent a lot of time on the C implementation not because it was hard, but because I wanted to experiment. I managed to turn the iterative algorithm into a single for loop, but realized that would probably hurt readability. Another interesting thing I managed to do in C was improve the recursive algorithm by creating a tail-call optimized version of it. This—along with -Ofast—actually made it just as fast as the iterative version of the algorithm. I didn't include it in this submission, though.
- **Ada**—this language was just as annoying as usual. Even without using strict compiler flags, gnat would always yell at me for the most mundane stuff. Using Long_Integer was also very annoying. But it works, and that's all that matters.
- **Fortran**—this was probably the worst out of the bunch. Weird function argument antics, strange bugs, and the arcane, archaic syntax bring me back to a time in January where I had to refactor a certain blasted Lucifer algorithm. No regrets, though. Fortran's SIMD instruction exploits as well as its strict aliasing rule make it a favourite among scholars and physicists to this day.

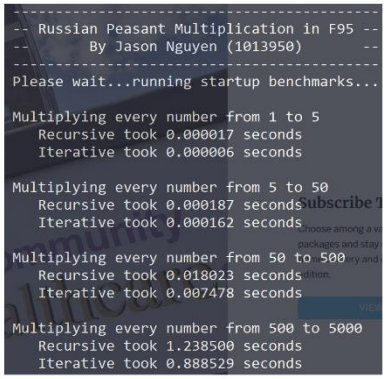
Efficiency wasn't at a huge contrast, but it was there.

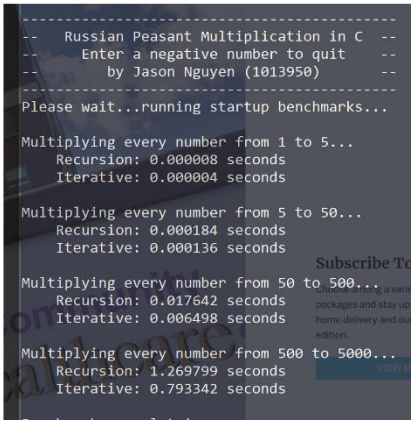
- Recursion consistently lost in every contest. All of the extraneous calls to the stack added so much extra overhead that when the 500-to-5000 tests commenced, the recursive implementations would sometimes take twice as long as their iterative counterparts!

- At very low numbers, there was almost no difference whatsoever. I attribute this to the fact that I didn't use UNIX's inaccurate timer and instead opted for intrinsic timer functions. As a result, there is almost no useful information in the 1-to-5 or 5-to-50 tests. Sometimes the recursive implementation would be exactly 0.00, sometimes the iterative would also be 0.00. Sometimes one would be 0.3, sometimes it would be the other. This is why I chose a lot of testing ranges.
    - 1 to 5 is pretty good for looking at the overhead of a program, because not much intrinsic work is done. This would be more useful if I used the UNIX built-in time feature, because it would really shine a light on the real-world latency involved.
    - 5 to 50 and 50 to 500 are both pretty in-between. They aren't very intensive, but they weren't paltry like 1 to 5 either.
    - 500 to 5000 was the main event to be honest. This actually took a decent amount of time, and in all tests, there is a double for loop (in order to make sure every possible selection of two numbers is chosen). That means 4500 * 4500 = 20 million number pairs were being processed.
        - **C** — **0.79s** vs. 1.27
        - **Ada** — 0.89s vs. 1.24
        - **Fortran** — 0.86s vs. **1.43**
    - As we can see above, the recursive Fortran implementation had the slowest runtime, and the iterative C implementation had the fastest. Through this, I think Ada is the slowest. Fortran had its speedy moments and C is a powerhouse of languages—but at the end of the day I think Ada is the most disadvantaged. It being a systems-critical, data-sensitive language based on being "correct" rather than "fast", Ada's paradigms clearly point to it sacrificing speed for the sake of lives. Whether it be avionics or missile control, Ada's very much far from the metal. Fortran and C were definitely the winners here as they have a closer connection to the assembly language they compile to. This leads to the next point.
    - Fortran and C are definitely more capable of dealing with big numbers than languages like Ada, Java, and other "high-level" implementations. This is because C and Fortran are very old languages with little abstraction from the machine code, and as a result, it is easier for the compiler to understand the user's intent. This means more optimized intermediate/machine code. You'll find C/Fortran programmers who resort to using bit shifts rather than stock multiplications as a result of this, and Fortran is widely revered for its uses in the math world (thanks to its single-instruction-multiple-data capabilities).

All in all, I enjoyed benchmarking these languages.


-- Russian Peasant Multiplication in Ada --
-- Enter a negative number to quit --
-- By Jason Nguyen (1013950) --
Please wait...running startup benchmarks...
Multiplying every number from 1 to 5
Recursive took 0.000003000 seconds
Iterative took 0.000003000 seconds
Multiplying every number from 5 to 50
Recursive took 0.000198000 seconds
Iterative took 0.000157000 seconds
Multiplying every number from 50 to 500
Recursive took 0.022470000 seconds
Iterative took 0.010114000 seconds
Multiplying every number from 500 to 5000
Recursive took 1.431143000 seconds
Iterative took 0.856655000 seconds


-- Russian Peasant Multiplication in F95 --
-- By Jason Nguyen (1013950) --
Please wait...running startup benchmarks...
Multiplying every number from 1 to 5
Recursive took 0.000017 seconds
Iterative took 0.000006 seconds
Multiplying every number from 5 to 50
Recursive took 0.000187 seconds
Iterative took 0.000162 seconds
Multiplying every number from 50 to 500
Recursive took 0.018023 seconds
Iterative took 0.007478 seconds
Multiplying every number from 500 to 5000
Recursive took 1.238500 seconds
Iterative took 0.888529 seconds


-- Russian Peasant Multiplication in C --
-- Enter a negative number to quit --
-- by Jason Nguyen (1013950) --
Please wait...running startup benchmarks...
Multiplying every number from 1 to 5...
Recursion: 0.000008 seconds
Iterative: 0.000004 seconds
Multiplying every number from 5 to 50...
Recursion: 0.000184 seconds
Iterative: 0.000136 seconds
Multiplying every number from 50 to 500...
Recursion: 0.017642 seconds
Iterative: 0.006498 seconds
Multiplying every number from 500 to 5000...
Recursion: 1.269799 seconds
Iterative: 0.793342 seconds