**Spike:** 24

**Title:** Performance Measurement

**Author:** Ben Holmes, 103024841

**Goals / deliverables:**

Demonstrate the following performance and measurement concepts:

1. Measurement: Show, with numbers and a chart, the performance of each provided function.
2. Scalability: Show, with numbers and a chart, how the performance of each function varies with different data sizes.
3. Repeatability: Show, with numbers and a chart, how repeatability will vary with multiple tests in a given run.
4. Compiler Settings: Modify your compiler optimisation settings and demonstrate, with numbers and a chart, a difference
5. Profiling: Use the Visual Studio performance profiler to profile the code. Provide a brief description of the results, including screenshots

**Technologies, Tools, and Resources used:**

* Visual studio
* Word
* Excel
* GitHub

**Tasks undertaken:**

* Download sample code
* Figure out how to run it
* Do each of the test types

**What we found out:**

All time axis are in microseconds

Measurement:

A screenshot of a computer

Description automatically generated

This is a single run with file size 100. Even with the small scale of this one, you can already see the difference between the functions speed which will persist throughout the tests.

Scalability:

This is a single run at each of the file sizes on the axis with the 4 values averaged. All 3 of the functions time taken does scale linearly with the file size.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Repeatability:

A 1000 size file was run against 5 times and all 20 results were stored and graphed

Simple:

Simple is quite inconsistent with its timing, with a lot of variation even in those near the trend line. Also potentially a problem, both large outsiders were the 4th run in there respective tests

Inside-Out

Inside-Out was much more consistent with only 3 massive outliers and the rest being similar

String Search:

String search had the most consistency, with the flattest of the trend lines.

A table with numbers and letters

Description automatically generated

Compiler

I ran a file size 1000 in debug mode, and then ran it in release mode



(changed at the debug dropdown)

A screenshot of a computer

Description automatically generated

For String search it cut down the lime taken to 10% of the original which was the lowest decrease of the 3.

Both simple loop and inside out dropped down to between 2-3% of the original.

A computer screen with white and yellow text

Description automatically generated

Profiling

This was a little tricky to sort out as I had to change a few of the project properties to get the instrumentation profiling working. Also this was run in debug mode with profiling, as I could not get release mode working , so the time taken is really high for a 1000 size file without profiling  
console output for proof/display:

A screenshot of a computer

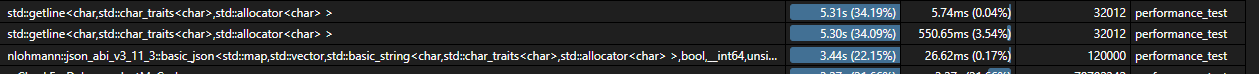
Description automatically generated

Time comparison between the 3 functions



this really shows the difference between the 3, with inside out being in a completely different league, especially under these conditions.

What I found most interesting is:



That the inside out and simple loop: Json [] operator (bottom of the 3, could not get it to show all) was called 4 times more than the getline function of the string search but used far less time, proving that the nhloman Json really is a fantastic file input and was the right choice for my zorkish stuff.

**A screenshot of a computer

Description automatically generated**