Priorities (in no particular order)

- **Time efficiency:** Our biggest priority is a programming language that performs the tasks we need efficiently.
- **Memory efficiency:** While we do not anticipate our tool to require an incredible amount of memory in order to run, we still want to consider the memory efficiency of the programming language that we choose to use.
- **Familiarity:** Of course, if we end up using a programming language we are unfamiliar with, there will be a significant learning curve in learning how to use the new programming language. Still, if the benefits of the language largely outweigh the negatives, we may consider learning a new language to make our tool.
- **Popularity:** On top of our familiarity, using a language that is popular will help to ensure that should other people want to work on this tool in the future, there is a good chance that they will already be familiar with the language the tool is coded in.
- **Readability**: Being able to understand the code written in the language at a glance is important both for the sake of ourselves, as well as any other person who may potentially work on the tool in the future.
- **Scalability:** The tool we are trying to make requires a lot of smaller parts that need to work together, therefore the language we choose should allow us to easily break our code down into smaller sections, across multiple files, as well as allowing us to use code from one file in another.
- **Portability:** We want to ensure that our tool is able to be used across a variety of platforms (mainly Windows, Mac, and Linux). The language we use should be able to compile to formats compatible with all of these platforms.
- **Functionality:** Due to the two semester time constraint of this project, it is important to be able to write working code in a timely fashion. Simple functions (e.g. file I/O, list manipulation) should be simple to implement in the chosen language.

Tests Used

- **Time efficiency:**
  - Variable updating: A variable, say x, is initialized with a value of zero, then is updated using "x = x + 1" until its value reaches one billion. The timer starts right before the variable is initialized and ends right after the variable reaches a value of one billion. Fifty trials are run, and the average completion time is recorded.
  - String array population: A string array of length one hundred thousand is instanced. Then, iterating through the array, each index is set to hold a string of length (index + 1). The timer starts right before the array is instanced, and ends right after the last value is written to the array. Fifty trails are run, and the average completion time is recorded.
  - Tree instancing: A binary tree with two hundred thousand nodes is created, each storing an integer. The timer starts right before the tree is instantiated, and ends when the tree is completely initialized. Fifty trials are run, and the average completion time is recorded.
  - Tree traversal: A binary tree with two million nodes is created, each storing an integer. Depth first search is performed to increment the data in each node. The timer starts right before the DFS is called, and ends right after the DFS function finishes its call. Fifty trials are run, and the average completion time is recorded.
- **Memory efficiency:**
  - Integer array: An integer array of size one million is created to hold the first million non-negative integers.
  - String array: A string array of size one hundred thousand is created, where the length of the string is equal to its index plus one.
  - Large tree: A binary tree with two hundred thousand nodes is created, each storing its parent reference, references to its left and right child, and a string of length one.

Notes:

- Source code for all tests run in all languages can be found here

C++

- **Time efficiency:**
  - Variable updating: 0.24 seconds
  - String array population: inconclusive due to issue with "max" array and string size (see notes)
  - Tree instancing: 0.0081 seconds
  - Tree traversal: 0.0069 seconds
- **Memory efficiency:**
  - Integer array: 4.3 MB
  - String array: inconclusive due to issue with "max" array and string size (see notes)
  - Large tree: 10.4 MB
- **Familiarity:** 5/10
- **Popularity:** 7/10
- **Readability**: 6/10
- **Scalability:** Very scalable
- **Portability:** Quite portable
- **Functionality:** 7/10

Notes:

- There seems to be an issue with C++ related to the maximum size of an array / string / object you can create in certain contexts. While there are certainly ways to get around this, there were no indicators of there being an issue when running the compiled code, which may cause a hassle if it happens again.

Java

- **Time efficiency:**
    - Variable updating: 0.80 seconds
    - String array population: 1.03 seconds
    - Tree instancing: 0.29 seconds
    - Tree traversal: 0.00541 seconds
- **Memory efficiency:**
    - Integer array: 26.2 MB
    - String array: 5.85GB
    - Large tree: 30.6 MB
- **Familiarity:** 10/10
- **Popularity:** 9/10
- **Readability**: 8/10
- **Scalability:** Very scalable
- **Portability:** Highly portable
- **Functionality:** 8/10

Python

- **Time efficiency:**
    - Variable updating: 37.2 seconds
    - String array population: 4.64 seconds
    - Tree instancing: 8.03 seconds
    - Tree traversal: 0.135 seconds
- **Memory efficiency:**
    - Integer array: 44.5 MB
    - String array: 4.78 GB
    - Large tree: 27.3 MB
- **Familiarity:** 8/10
- **Popularity:** 10/10
- **Readability**: 9/10
- **Scalability:** Scalable, but with caveats
- **Portability:** Highly portable
- **Functionality:** 10/10

Rust

- **Time efficiency:**
    - Variable updating: 0.40 seconds
    - String array population: 8.83 seconds
    - Tree instancing:
    - Tree traversal:
- **Memory efficiency:**
    - Integer array: 4.1 MB
    - String array: 5.29 GB
    - Large tree:
- **Familiarity:** 3/10
- **Popularity:** 4/10
- **Readability**: 5/10
- **Scalability:** Scalable, but with caveats
- **Portability:** Quite portable
- **Functionality:** 7/10

	C++	Java	Python	Rust
Variable updating	0.24 s	0.80 s	37.2 s	0.40 s
String array population	N/A	1.03 s	4.64 s	8.83 s
Tree instancing	0.0081 s	0.29 s	8.03 s	Not found
Tree traversal	0.0069 s	0.00521 s	0.135 s	Not found
Integer array	4.3 MB	26.2 MB	44.5 MB	4.1 MB
String array	N/A	5.85 GB	4.78 GB	5.29 GB
Large tree	10.4 MB	30.6 MB	27.3 MB	Not found