Academic Reference:

We will be implementing the KMP algorithm for our final project. Our academic reference is: https://epubs.siam.org/doi/10.1137/0206024. Along with this, we will be using the wikipedia definition of this algorithm to guide us in our process: https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm.

Algorithm Summary:

The KMP algorithm is a very efficient pattern matching technique to find occurrences of a pattern in a larger text. It does this by skipping unnecessary character comparisons. There are two main steps in the primary version of this article. The first step is called preprocessing and has a prefix table based on the pattern. The table is called an LPS array and stores information about match positions. The second step is where the actual search happens. It iterates through the text while using the LPS array. The algorithm then returns the location if it finds a match for the pattern. It is supposed to run in O(m + n) time.

Function Input/Output:

The input for our function is a string that we want to search for occurrences of our pattern along with the actual pattern that we are looking for. The output will be a vector of integers that detail where the patterns are found in the text.
Our first function is meant to read the txt files. We will use <sstream> to do so.
Our second function we plan to implement is the preprocess function:

- void preprocessLPS(const std::string& pattern, std::vector<int>& lps)

This preprocess function is a helper function that produces the LPS vector. The input will be the pattern that we are analyzing and the vector we will change and then return. The first two tests check to make sure that the table is correctly populated. We will do this with some “practice” text in our tests.
Our third function is the main KMP algorithm:

- std::vector<int> searchKMP(const std::string& text, const std::string& pattern)

This function takes the text we are searching and the pattern as its inputs. It then performs the main KMP algorithm. Our third test checks to make sure that this function correctly returns the correct number of patterns. Specifically, on our small test case of Percy.txt, the length of the input for the pattern “half-blood” should be 2.

Data Description:
Our data consists of excerpts from some works of literature from some books that we both like. It is found in the /data folder of our repository and we will use it to test the correctness of our algorithm. We will have to write the method to read the txt files as well.