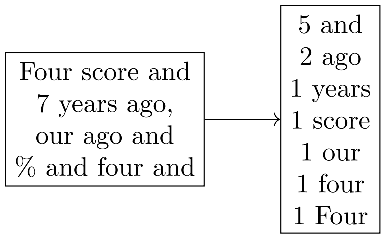
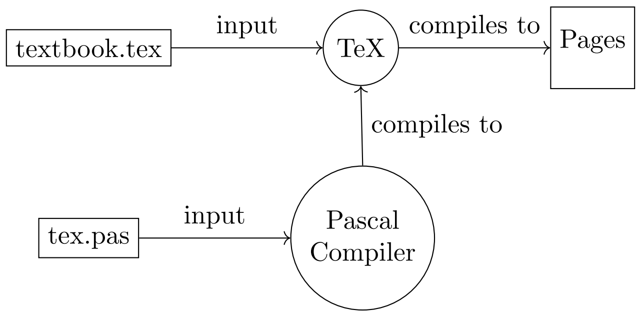
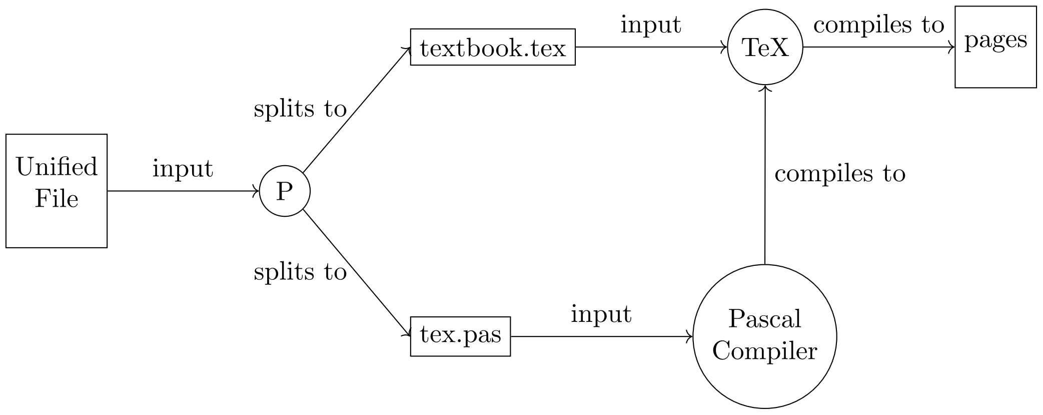
We will begin this class with a quiz. We wish to develop a program that takes a sequence of ASCII characters and outputs a frequency-sorted concordance of all words in the input.
We define a word with the following regular expression: [A-Za-z]+
Words must be bounded by white space. An example usage of our program is:



This problem was used by Turing Award winner DE Knuth as press for his book, The Art of Computer Programming
The motivation came from difficulties in creating the initial press for his book;
  The lead type press could not handle code or equations, which were fundamental!
So Knuth invented a language to produce correct images, called Tex.
He developed a textbook to bring attention to his language, which explained the framework. The framework took the form:



This presents a large (and common) issue – code and documentation must be kept in agreement.
Knuth developed the idea of a unified, interleaved file to combat this.
Thus his program took the following final form:



This left him with a nearly 500 page textbook; how would he drum up interest?
Like a good computer scientist, he chose to write a paper. He coined his approach **literate programming**.
This has become standard practice in many disciplines; we can see JavaLibrary for proof.

Knuth approached Doug McIlroy, the manager for the development of UNIX, for help.
McIlroy suggested our problem as an example
Knuth wrote a solution using Hash Tries via literate programming!
Running it through TeX gives pages and through pascal gives the paper!

As impressive as this was, the afterword was what stuck.
McIlroy proposed the following trivial BASH solution (he was the brain behind UNIX pipes after all)

```
tr -c 'A-Za-z' '\n' | sort | uniq -c | sort -rn
```

The pascal solution was, say, 1000 lines.
    It was faster
    It was more "checked" by compilers
BUT the bash solution is exceedingly simple and documents itself as follows:

$$\begin{tikzcd}
\boxed{\begin{array}{l}\text{Four}\\ \text{Score}\\ \text{and 7}\\ \text{years:.}\end{array}} \arrow[r, "\text{tr}"] & \boxed{\begin{array}{l}\text{Four}\\ \text{Score}\\ \text{and}\\ \text{years:.}\end{array}} \arrow[r, "\text{sort}"] & \boxed{\begin{array}{l}\text{Four}\\ \text{ago}\\ \text{and}\\ \text{score:.}\end{array}} \arrow[r, "\text{uniq}"] & \boxed{\begin{array}{l}\text{1 Four}\\ \text{2 ago}\\ \text{4 and}\\ \text{1 score}\\ \text{...}\end{array}} \arrow[r, "\text{sort}"] & \boxed{\begin{array}{l}\text{4 and}\\ \text{2 ago}\\ \text{1 Four}\\ \text{1 score}\\ \text{...}\end{array}}
\end{tikzcd}$$

This demonstrates an issue fundamental to computer science called **choice of notation**.
There are pros and cons to any choice of notation, and it is up to the engineer to weigh them!
As this is a discussion of languages, consider the following fundamental linguistic hypothesis:

Sapir-Whorf Hypothesis.

a) There is no limit on the structural diversity of language.

b) The structure of a language determines a native speaker's perception of experience.

This was proposed by linguists for natural languages, but (b) was softened.
Even though it is false, it stuck: see "Eskimos have 19 words for snow".
    Even if false for natural languages, it is true for programming languages!
We explore this in the next lecture