Literate Programming

- Instead of the source code and documentation in two places, put them together to be useful on their own

Problem at the start of class:

- Given an input stream, gather the number of occurrences of each word and then rank them in decreasing order

Shell solution:

```
tr -cs `A-Za-z` `[\n]` | srt | uniq -c | sort -rn
```

Language Design Issues (language "wars")

- Wars are often between neighbors
- How do we decide which language to use?
  - How do we measure efficiency at runtime?
    - power(W), time(s), joules(J), space(β), network access, utilization(%), (real time, CPU time), ecosystem size, libraries, tools
  - ecosystem size, libraries, tools
  - syntax issues
    - simplicity
    - readability
    - compatibility/interoperability
    - documentation/learnability

Orthogonality

- One of the language features doesn't affect the choices you make
  - ex. You can't return an array from a function in C/C++

Safety

- How safe is your language in the presence of the bugs you are going to have?
  - ex. Undefined behavior

```
char *p = nullptr;
...
return *p;
```

Abstraction

- We don't want to always deal with low level processes
  - e.g. functions, classes, packages, modules