## Pipeline Architecture

 $Software\ Architecture$ 

Brae Webb

February 26, 2024

## So far...

Layered architectures reduce the impact of changing a layer

## Question

Why does the layer order matter?

## Question

Why does the layer order matter?

Answer

Each layer implements a different interface.

## So...

If every layer implements the same interface?

## $Extreme\ layered\ architecture$

## Pipeline Architectures<sup>1</sup>

## Definition 1. Pipeline Architecture

Components connected in such a way that the output of one component is the input of another.

## Question

Can you think of a *pipeline archiecture*?

## Question

Can you think of a *pipeline archiecture*?

Answer

How about *bash*?

```
>> cat assignment.py | grep "hack" | wc -l | tee code-quality.txt
```

#### Notice:

• Each program performs a small well-defined task.

#### Notice:

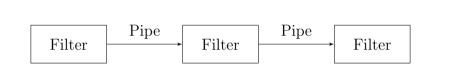
- Each program performs a small well-defined task.
- Each program implements the same interface (i.e. raw text).

wc -l

→ tee code-quality.txt

grep "hack"

cat assignment.py





## Filters

Modular software components



#### Filters

Modular software components

## Pipes

The flow of data between filters

 ${\bf Producers}$ 

Source of data

Producers

Source of data

Transformers

Transform data

Producers

Source of data

Transformers

Transform data

Testers

Filter data

| Producers      | Testers                 |
|----------------|-------------------------|
| Source of data | $\mathrm{Filt}\epsilon$ |

data Filter data

5

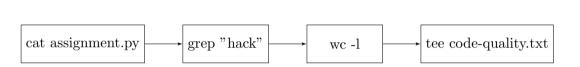
Consumers

Transformers
Transform data

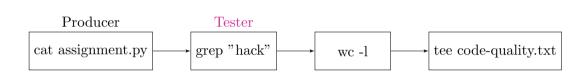
Target for results

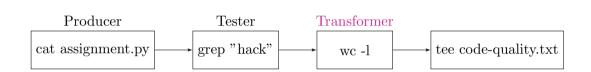
#### Exercise

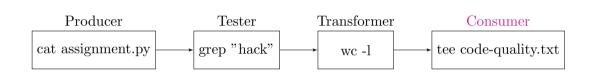
Label the bash pipeline



## Producer







## Definition 2. One Direction Principle

Data should flow in one direction — downstream.

## Definition 3. Independent Filter Principle

Filters should not rely on specific upstream or downstream components.

## Corollary 1. Generic Interface

The interface between filters should be generic.

## Corollary 2. Composable Filters

Filters (i.e. Transformers & Testers) can be applied in any order.

The Case Study
Bash

#### POSTX-2017

#### Shell & Utilities

#### Utilities

admin - create and administer SCCS files (DEVELOPMENT) ex - text editor expand - convert tabs to spaces alias - define or display aliases ar - create and maintain library archives expr - evaluate arguments as an expression asa - interpret carriage-control characters false - return false value at - execute commands at a later time fc - process the command history list awk - pattern scanning and processing language fg - run jobs in the foreground file - determine file type basename - return non-directory portion of a pathname batch - schedule commands to be executed in a batch queue find . find files fold . filter for folding lines bc - arbitrary-precision arithmetic language bg - run jobs in the background fort77 - FORTRAN compiler (FORTRAN) c99 - compile standard C programs fuser - list process IDs of all processes that have one or more files cal - print a calendar cat - concatenate and print files gencat - generate a formatted message catalog cd - change the working directory getconf - get configuration values cflow - generate a C-language flowgraph (DEVELOPHENT) get - get a version of an SCCS file (DEVELOPMENT) charp - change the file group ownership getopts - parse utility options chand - change the file modes gren - search a file for a pattern chown - change the file ownership hash - remember or report utility locations cksum . write file checksums and sizes head - copy the first part of files cmp - compare two files icony - codeset conversion command - execute a simple command id - return user identity comm - select or reject lines common to two files ipcrm - remove an XSI message queue, semaphore set, or shared memory compress - compress data segment identifier incs - report XSI interprocess communication facilities status cp - copy files jobs - display status of jobs in the current session crontab - schedule periodic background work ioin - relational database operator ceplit - enlit files based on context ctags - create a tags file (DEVELOPMENT, FORTRAN) kill - terminate or signal processes cut - cut out selected fields of each line of a file lex - generate programs for lexical tasks (DEVELOPMENT) link - call link function cycef - generate a C-language program cross-reference table In - link files (DEVELOPMENT) localedef - define locale environment date - write the date and time dd - convert and copy a file locale - get locale-specific information logger - log messages delta - make a delta (change) to an SCCS file (DEVELOPMENT) logname - return the user's login name df - report free disk space diff - compare two files lp - send files to a printer dirname - return the directory portion of a pathname ls - list directory contents du - estimate file space usage m4 - macro processor echo - write arguments to standard output mailx - process messages ed - edit text make - maintain, update, and regenerate groups of programs env - set the environment for command invocation (DEVELOPHENT)

## Question

Who has heard of *literate programming*?

## The Challenge — set by Jon Bently

- 1. Read a file of text.
- 2. Determine the n most frequently used words.
- 3. Print out a sorted list of those words along with their frequencies.

## Knuth's Solution

17 pages of elegant and descriptive code.

hy lon Bentley with Special Guest Ousters Don Knuth and Doug Mclirou

# programming pearls

#### A LITERATE PROGRAM

Last monthly column introduced from Funthly style of "Literate Programming" and his WEB system for building programs that are works of literature. This column precents a literate program by Vouth fits prining ore sketched in last month's column) and, as befits literature, a review. So without further ado, here is Knuth's progress retypeset in Communications style.

| Common Words             |  |  |  |  |  |  |  |  |  |  |  |  | Section |  |   |   |  |  |  |  |    |  |
|--------------------------|--|--|--|--|--|--|--|--|--|--|--|--|---------|--|---|---|--|--|--|--|----|--|
| Introduction             |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   |   |  |  |  |  |    |  |
| Strategic considerations |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   |   |  |  |  |  | 8  |  |
| Basic input routines     |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   |   |  |  |  |  | 9  |  |
| Dictionary lookup        |  |  |  |  |  |  |  |  |  |  |  |  |         |  | , |   |  |  |  |  | 17 |  |
| The frequency counts .   |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   |   |  |  |  |  | 32 |  |
| Sorting a trie           |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   |   |  |  |  |  | 36 |  |
| The endgame              |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   |   |  |  |  |  | 41 |  |
| Index                    |  |  |  |  |  |  |  |  |  |  |  |  |         |  |   | , |  |  |  |  | 42 |  |

- 1. Introduction. The purpose of this program is to solve the following problem posed by Ion Bentley:
- Given a text file and an integer k, print the k most common words in the file (and the number of their occurrences) in decreasing frequency.
- Ion intentionally left the problem community vague, but he stated that "a user should be able to find the 100 most frequent words in a twenty-page technical naper (muchly a 50K byte file) without undue emotional trauma
- Let us agree that a word is a sequence of one or more continuous letters: "Bent Lev" is a word, but "ain't" isn't. The sequence of letters should be maximal, in the cence that it cannot be lengthened without including a nonletter. Uppercase letters are considered equivalent to their lowercase counterparts, so that the words "Bentley" and \*DERTLEY" and "bent lov" are exentially identical.
- The given problem still isn't well defined, for the file might contain more than & words, all of the same

fractioners or there might not even be as many as k words. Let's be more precise: The most common woods are to be printed in order of decreasing frequarter with words of aqual fraquency listed in alphabetic order. Printing should stop after k words have been output, if more than it words are present.

- 2. The input file is assumed to contain the given text. If it begins with a positive decimal number (preceded by optional blanks), that number will be the value of k otherwise we shall assume that k = 100. Answers will be sent to the output file.
- define default it as 100 lose this value if it isn't otherwise specified!
- 3. Bestdes solving the given problem, this program is supposed to be an example of the WEB system, for noonle who know some Parcal but who have never seen NEB before. Here is an outline of the program to be constructed:
- program common mords (input, output): type (Type declarations 17) var (Global variables 4) (Procedures for initialization s) (Procedures for input and output a) (Procedures for data manipulation 20) begin (The main program a):
- 4. The main idea of the WEB approach is to let the program grow in natural stages, with its parts preconted in roughly the order that they might have been written by a programmer who isn't consciolly clairvovant
- For example, each clobal variable will be introduced when we first know that it is necessary or desirable: the WEB system will take care of collecting these declarations into the proper place. We already know about one global variable, namely the number that Bantley called k Let us give it the more descriptive name may words to print.

Programming Pearls

(Global variables 4) = may mards to print integer

See also sections 11, 13, 18, 22, 32, and 36,

a. As we introduce new global variables, we'll often want to give them certain starting values. This will be done by the initialize procedure, whose body will consist of various pieces of code to be specified when we think of particular kinds of initialization.

lat most this many words will be printed!

- (Procedures for initialization 5) = or or miles or destriction
- var i: intever: fall-purpose index for initializa-
- begin (Set initial values 12)
- This rade is used in section 3.
- 6. The WEB system, which may be thought of as a preprocessor for Pascal, includes a macro definition facility so that portable programs are easier to write. For example, we have already defined 'default\_k' to be 100. Here are two more examples of MEB macros: they allow us to write, e.g., 'incricount[ v])' as a convenient abbreviation for the statement 'coverful ecount[v] + 1'
  - define  $incr(\#) = \# \leftarrow \# + 1$  (increment a varidefine  $decr(w) \equiv w \leftarrow w - 1$  (decrement a vari-
- 7. Some of the procedures we shall be writing come to abrunt conclusions; hence it will be convenient to introduce a 'return' macro for the operation of jumping to the end of the procedure. A symbolic label 'exit' will be declared in all such procedures. and 'exits' will be placed just before the final and (No other labels or sate statements are used in the present program, but the author would find it pain. ful to eliminate these particular must)
- define exit = 30. (the end of a procedure) define return = goto exit | lquick termination|
- 8. Strategic considerations. What algorithms and data atmostrator about discussed for Doutlands area lem? Clearly we need to be able to recognize different occurrences of the same word, so some sort of internal dictionary is necessary. There's no obvious way to decide that a particular word of the input cannot possibly be in the final set, until we've gotten very near the end of the file; so we might as well remember every word that appears

There should be a frequency count associated with each word, and we will eventually want to run through the words in order of decreasing frequency. But there's no need to keen these counts in order as we read through the input since the order matters only at the and

Therefore it makes sense to structure our program as follows:

(The main program s) =

initialize-(Establish the value of our analy to oriet usy

(Input the text, maintaining a dictionary with frequency counts 2():

(Sort the dictionary by frequency 39): (Output the results 41)

This code is used in partion 3

9 Resic input routines | Let's switch to a hottom. up approach now by writing some of the procedures

that we know will be necessary sooner or later. Then we'll have some confidence that our program is taking shane, even though we haven't decided yet how to handle the searching or the sorting. It will be nice to get the messy details of Pascal input out of the way and off our minds

Here's a function that reads an optional positive integer, returning zero if none is present at the besinning of the current line

(Procedures for input and output a) = function read into integers

var v: integer: Ithe accumulated value! booin v - O if not then

begin while  $(\neg cots) \land (insut \uparrow = '...')$  do ortlinnuth while (input t > '0') A (input t < '9') do begin it an 10 an a confirmation and 10 to

get(input); end.

end: read int - m end:

See also sections 15, 35, and 40. This code is used in section 2

10. We invoke reed intonly once.

(Establish the value of max words to print to) = max\_words\_to\_print - read\_int: If may monte to print = 0 then  $max_words_to_print \leftarrow default_k$ 

This code is used in section 8.

11. To find words in the innut file, we want a quick way to distinguish letters from nonletters. Pascal has

#### McIlroy's Solution

```
tr -cs A-Za-z '\n' | \
    tr A-Z a-z | \
    sort | \
    uniq -c | \
    sort -rn | \
    sed ${1}q
```

## Question

Is literate programming bad?

## Question

Is literate programming bad?

Answer

No, the Unix philosophy is just good.

## The Unix Philosophy

• Write programs that do one thing and do it well.

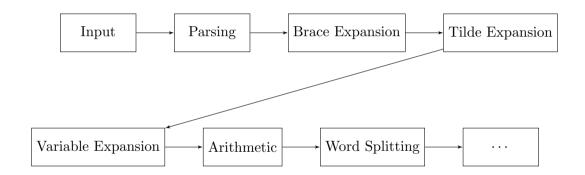
## The Unix Philosophy

- Write programs that do one thing and do it well.
- Write programs to work together.

## The Unix Philosophy

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

## Bash itself is a pipeline



## Reading...

"Pipeline Architecture" Notes [Webb and Thomas, 2023]

#### References

[Webb and Thomas, 2023] Webb, B. and Thomas, R. (2023). Pipeline architecture.

https://csse6400.uqcloud.net/handouts/pipeline.pdf.