Design

Is design important? 75%-80% of system errors are created in the analysis and design phases.

Analysis and design phases account for about only 10% of the overall system cost.

Only about 25% of software projects result in working systems.
(Perhaps you get what you pay for.)

WWMCCS network (Nov 9, 1979, potential global thermonuclear war)[1]
(human error loaded training tape into active system)

Therac-25 Medical Linear Accelerator (1985-1987, >= 3 deaths) [2]
(software safety controls, hardware interlocks, testing failures)

Patriot Anti-missile Timing Bug (1991, 28 deaths, 98 wounded) [3]
(concurrency bugs, numeric precision limitations)

Analyzing the Requirements

You have a very detailed specification for the GIS system.

That's actually very different from a typical experience in a the workplace.

There, you're likely to have a client with:

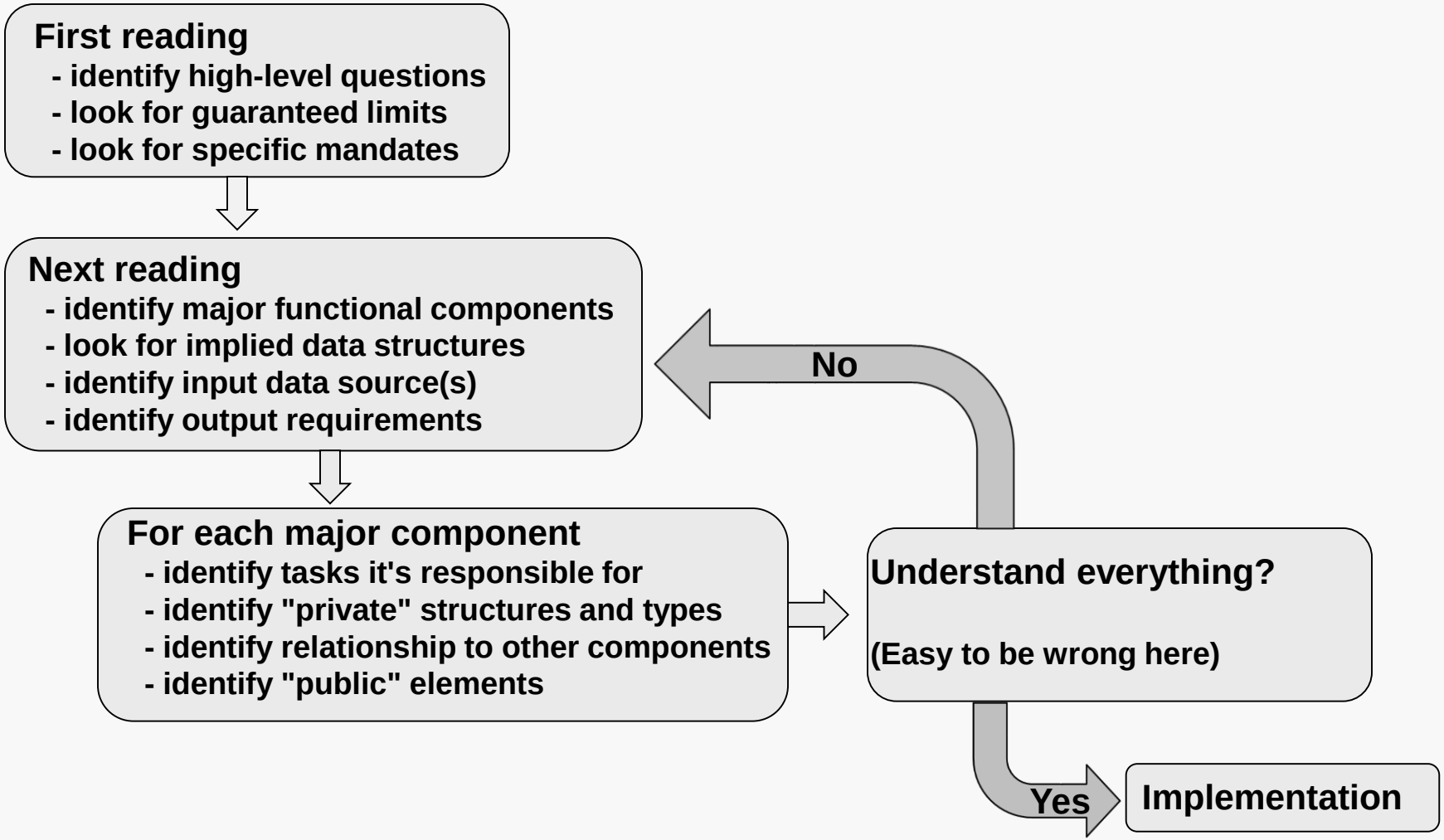
- some vague notions of what is actually needed (or desired)
- little ability to clearly express those vague notions
- no understanding of the limitations imposed by hardware and language
- solid protections against the more effective interrogation techniques

Here, you have:

- lots of details regarding what is required
- test data and reference output to use

Analyzing the Requirements

I'd say the right way to approach the specification is something like this:



Limitations

The rest of this presentation shows ONE approach to analyzing a particular specification.

It is certainly not the only way.

It is also not a complete rendition of the design I used, but it does imply the rest.

The discussion leaves some useful questions unidentified, and therefore unasked.

It does not discuss coding issues at all… they generally have absolutely nothing to do with the creation of a design.

Identifying Specific Mandates and Limits

Here are a few that are easily found in the specification:

Must be able to deal with a specified set of search queries.

Must achieve a certain efficiency in dealing with queries.

Must use arrays for data structures.

Never need to deal with more than a specified number of data records.

Length limits on certain record fields.

Identifying Major Components

It helps to visualize the system and imagine it in operation.

dB file (file of GIS records)
script file (file of commands to be serviced)
log file (file holding results of command servicing)

command processor (uses other components to execute commands)

feature name index (map feature name + state abbreviation to dB file offset)

feature ID index (map feature ID to dB file offset)

Identifying Major Components

Sometimes the analysis of a components reveals the need (or benefit) of including more subtle compenents:

command processor (uses other components to execute commands)

index (provides interface for index lookups, encapsulates lower-level indices)

feature name index (map feature name + state abbreviation to dB file offset)

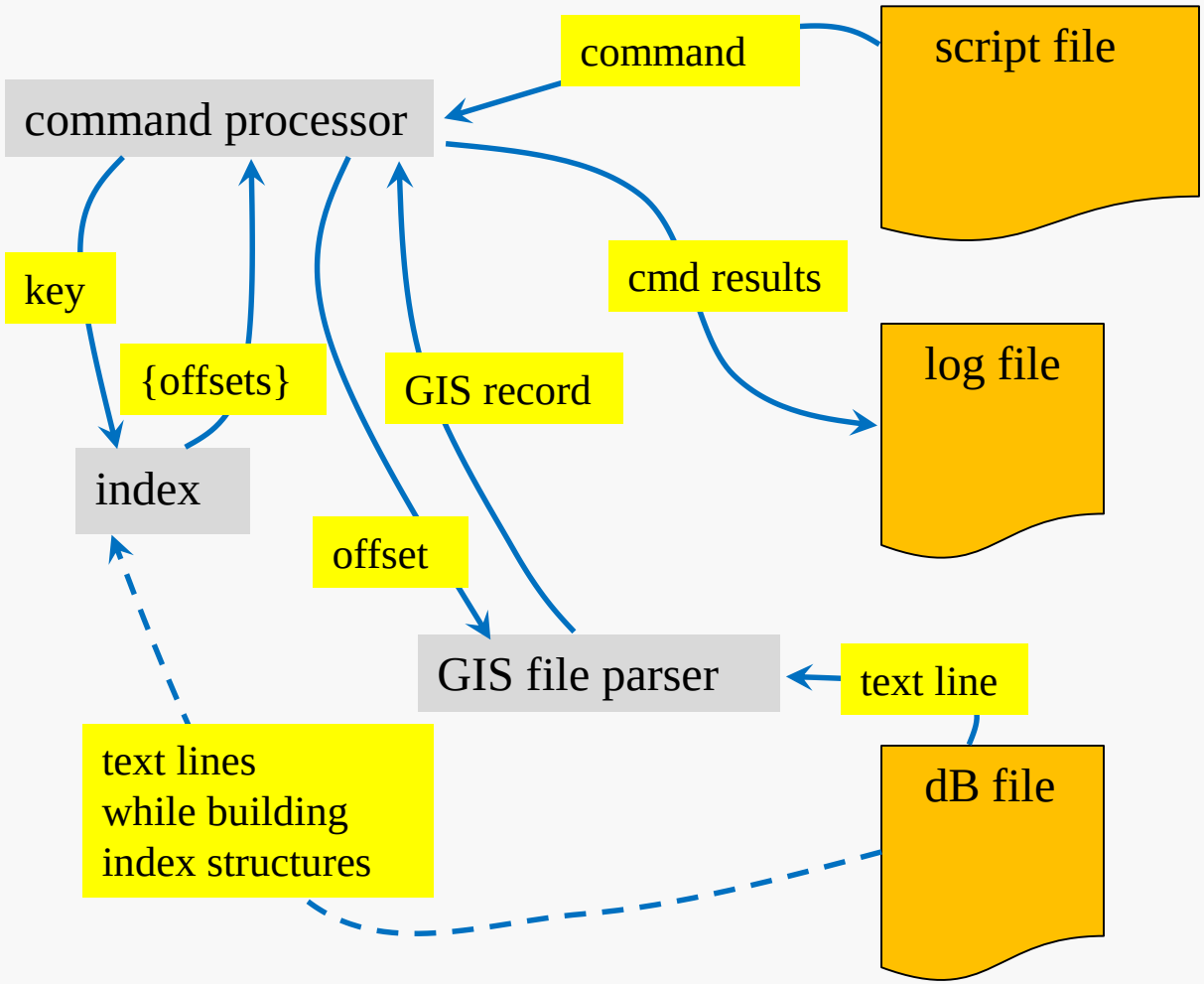
feature ID index (map feature ID to dB file offset)

GIS file parser (front end for accesses to the dB file)

- isolates handling formatting of GIS records

Identifying Relationships

We need to understand which components interact, and what communication takes place.



Identifying Relationships

Now, here's how I envision the system at work:

Phase 0:

Verify input files exist

Phase 1:

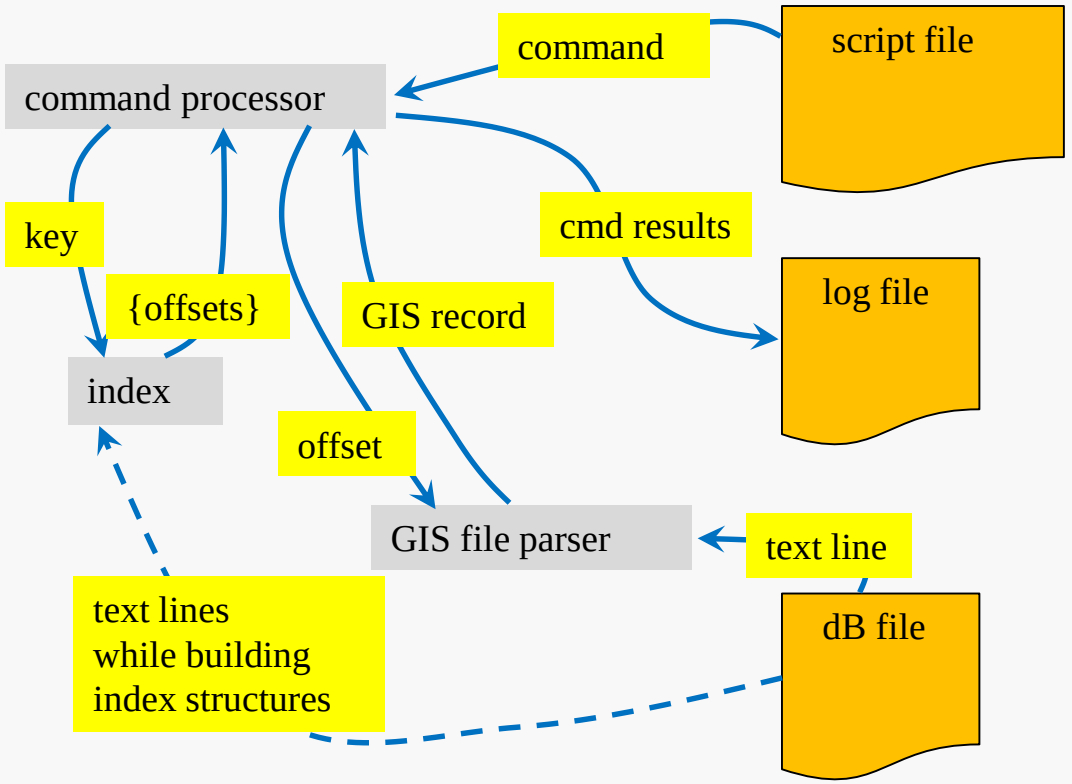
Build Name index and Feature ID index

Phase 2:

Process commands from script file

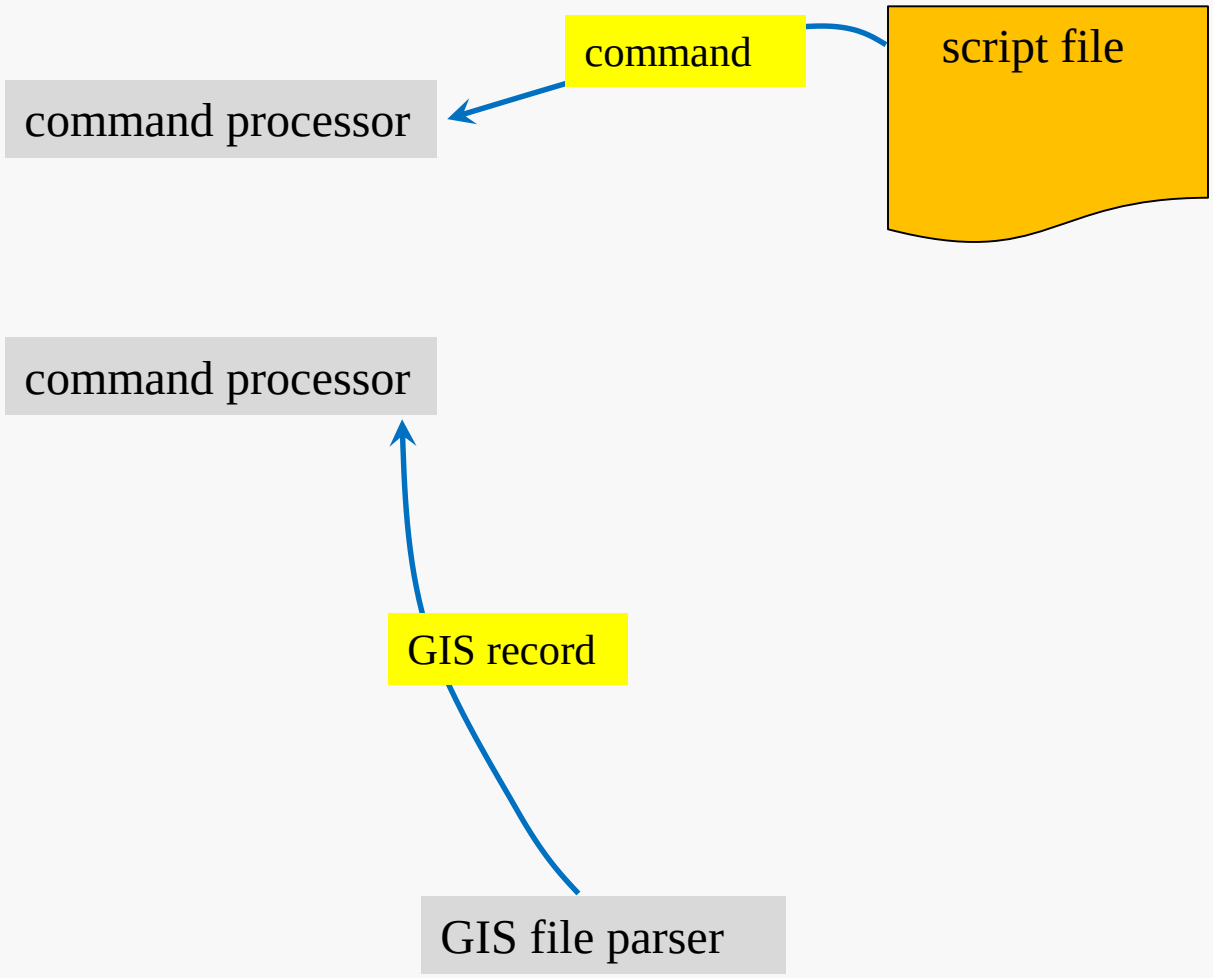
Phase 3:

Close files, deallocate memory, shut down



Identifying Data Types

Sometimes the analysis of relationships reveals the possible benefit of additional user-defined data types:



Identifying Data Types

A component may require internal types, that would not be meaningful to other components:

feature name index (map feature name + state abbreviation to dB file offset)

array of objects holding:
- a key value
- set of record offsets

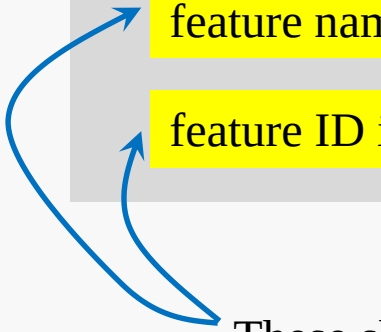
index entry object:
- key: depends on feature name and state abbrev?
- how to store a set of record offsets?

Identifying Private Elements of a Component

A component may require internal parts, that should not be accessible to other components:

index (provides interface for index lookups, encapsulates lower-level indices)

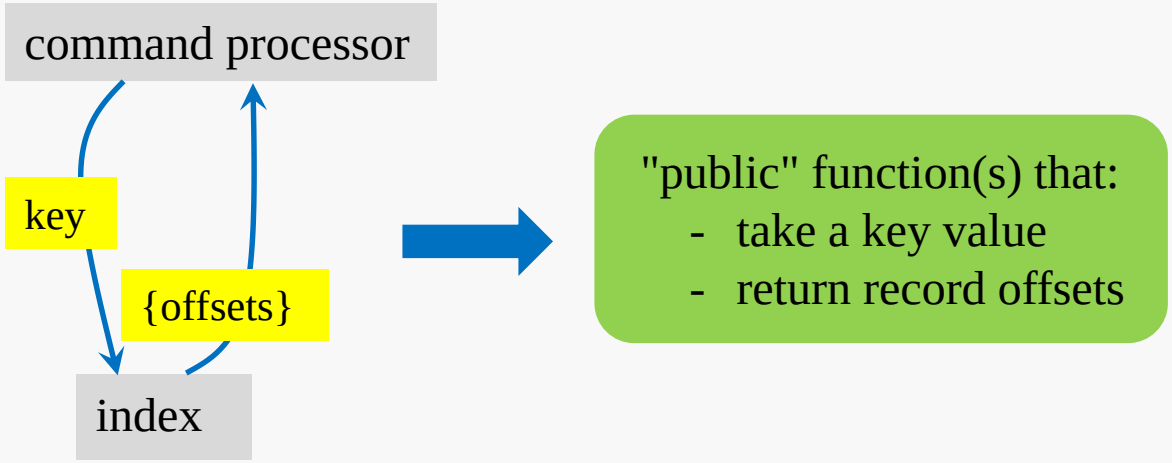
- feature name index (map feature name + state abbreviation to dB file offset)
- feature ID index (map feature ID to dB file offset)



These should NOT be accessed directly by other components.

Identifying Public Elements of a Component

Each interaction requires at least one "public" function to support it:



Packaging Code into Modules

A *module* is essentially a way of packaging the types, structures, and functions needed to provide a certain aspect of functionality.

Generally, I would consider each component identified earlier to be a potential module, leading to a C source file and an accompanying C header file.

Triple-check Everything!

Probably, no design is perfect.

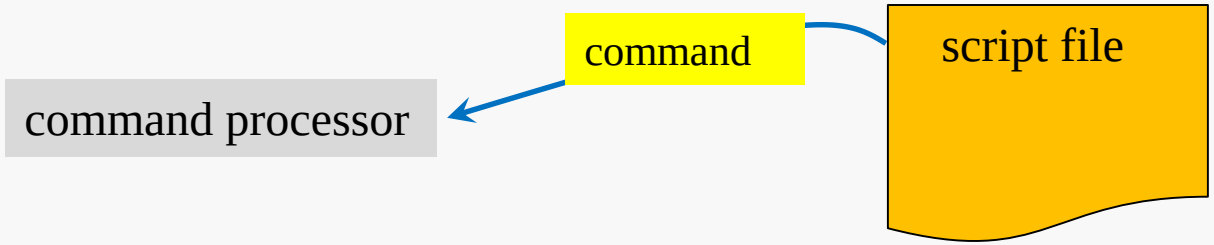
Probably, every design could be improved.

- By adding something overlooked earlier.
- By removing something identified earlier..

But, eventually we need to move on to creating an implementation.

- Frequently you will discover design flaws during the implementation phase.
- That can be painful…
- … which is just another reason careful attention to design pays off.

Thinking Implementation



Does a Command type make sense?

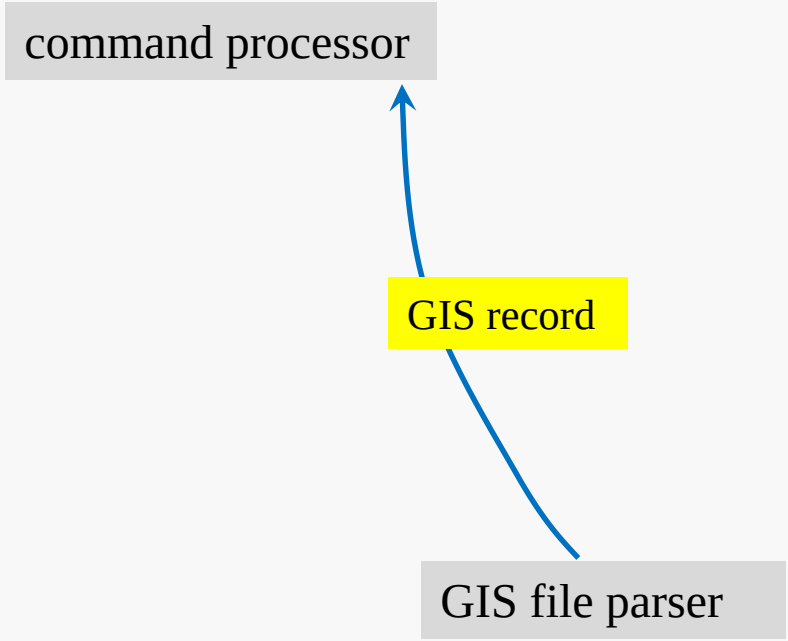
Does having a Command type make anything simpler or more efficient?

What entity would create a command object?

Should there be another component in the system design to do this?

All good questions… something to consider…

Thinking Implementation



Does having a GIS record type make sense?

Does having a GIS record type make anything simpler or more efficient?

Thinking Implementation

How am I going to acquire the different GIS record fields?

```
1481851|Blowing Springs Campground|Locale|VA|51|Bath|017|380408N|0795304W|. . .
```

One option:

Grab the whole line as a string: `fgets()`

Parse the string into fields: `sscanf()` or `strtok()` or ??

Another option:

Grab the fields one by one from the file: `fscanf()` or ??

c04: parsing strings

Thinking Implementation

How am I going to deal with file offsets?

Some useful functions:

`ftell()`

`fseek()`

Look at what's available in `stdio.h` (pubs.opengroup.org)

Implementation: Getting Started

Where do you start?

- getting command-line parameters
- verifying script file exists (exit usefully if not)
- getting name of GIS file and hash table size from script file
- verifying GIS file exists (exit usefully if not)
- creating log file

Each step is simple, essential, and easy to test.

Should be thinking about code organization (modules) already.
If all of this is in one file, you are probably not thinking well.

Implementation: the Index Structures

What comes next?

- building the Feature Name index
- building the FID index

Issues to consider:

- are you using your c07 solution or the reference hash table?
- are you going to build the FID index array in sorted order, or build it and then sort it?
- should you build both indices in one pass through the GIS file (more efficient), or in two passes (perhaps simpler to think about)?

Implementation: Modules and Interfaces

How many modules do you have now?

- one for the FID index and one for the Feature Name index?
- or a single module for both?
- or only one monolithic module for the whole assignment?

If your answer is the third option, I have grave doubts about your future.

What is the "public" interface of each module?

- function(s) to create empty index structures?
- for data structures: insertion and search functions (deletion is not needed for this assignment)
- function(s) to clean up the dynamic allocations used in each index?

Implementation: Handling the Commands

How are you going to deal with the command script?

- use a "command handler" to iterate through the commands?
- use a dedicated parser to retrieve the commands from the script?
- use a dedicated "handler" for each type of search command?

What do these considerations say about additional modules in your design or your implementation?

Implementation: Handling the Commands

Implement command handling one piece at a time:

- verify that you can retrieve the commands from the script (grab and print to stdout)
- verify that you handle comment lines in the script file correctly
- write code to handle one particular kind of search command and test that
- take advantage of the fact that you can comment out the commands you're not ready to test (or just write primitive handler stubs that say "I don't do that yet")

One key to producing a complex system is to be *incremental*: implement and test one feature at a time.

Another key is to retest "old" features after new ones are added… be sure that changes have not broken something that was previously working.

Sources

[1] *Close Calls with Nuclear Weapons*, Union of Concerned Scientists
www.ucsusa.org/sites/default/files/attach/2015/04/Close%20Calls%20with%20Nuclear%20Weapons.pdf

Command and Control: Nuclear Weapons, the Damascus Accident, and the Illusion of Safety, Eric Schlosser, Penguin Books, 2014 (978-0143125785)

[2] *An Investigation of the Therac-25 Accidents*, Nancy G Levenson, University of Washington and Clark S Turner, University of California, Irvine.
ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=274940

[3] *Patriot Missile Defense: Software Problem Led to System Failure at Dharan, Saudi Arabia*, IMTEC-92-26
www.gao.gov/products/IMTEC-92-26