debugging workshop

wahooooooo

good practices

The Employer-Employee Analogy

- You are a director of project
- You can employ as many employees as you want
- You have to manage all
- You have to direct all
- You have to evaluate all

Identify Subtasks

- How specific vs general
- Input ←
- Output →
- Subtask flow
    - how subtasks connect

Question: What are subtasks in mdb-add?

mdb-add-cs3157

- This is a program that inserts a record into the mdb-cs3157 database file. It will ask for a name and a short message, and then fills up the following structure with them:

```
struct MdbRec {
    char name[16];
    char  msg[24];
};
```

The structure in memory is added to the in memory database, and written at the end of the database file. It will confirm the addition by printing out the record identically to mdb-lookup. Note that the name and the message will be truncated to 15 and 23 characters, respectively, in order to fit them into the structure.

mdb-add

1. Load database into memory
    - Input? Output?
2. Get record data from user
    - Sub-subtasks (e.g. Prompting, Truncating)
3. Write data to memory
4. Write data to database
5. Print out data**
6. Clean up

Fit the Pieces

- LEGO Analogy
- Input prep
- `fopen` before calling `loadmdb`

Check Subtask Flow

- Sum of subtasks = the target executable
- Double-check specs

Review Subtasks

- Neatly Packed
  - Employee analogy
  - Keep evaluation in mind

Time to Code! wahoo

- Do I need functions?
  - Frequency of use
  - Magnitude of project
  - Testing style
  - Functions - Signature & Implementation (Input/Output)
- Working from pre-existing codebase
  - Take time to understand the code
  - **Carefully select** only the parts you need
- Diagram (lab 3)
- Pseudocode

Work-Flow

- Code **iteratively -** DO **NOT** code entire thing at once!
- Go by sub-task
  - Sum of mini-programs = big program
- Test each sub-task
  - `make` your Makefile first!
- Commit each sub-task
  - Checking each box
  - Reliable record

testing

Within Code

- Print statements and assert statements
- Comment out parts
- Tweaking inputs/parameters
- Breakpoint Pros
  - Simpler/Less Work (Ideal for small codebase)
- Breakpoint Cons
  - Can get messy if code too large
  - Commenting out might not always be possible
  - Changing code is too risky

External Tests

1. Isolating cases
   - EVERYTHING LOOKS RIGHT but it doesn't work
   - Replicate particular parts in different files
     - Function calls/Various statements
     - Start a fresh file, build iteratively until you recreate error
2. Test Driver
   - Remember Lab 3?
   - Ideal for code where most things are in functions
   - Update as you finish each function!
   - Question: How would you test `loadmdb`

```
int loadmdb(FILE *fp, struct List *dest)
```

Identifying Edge Cases

Think about:

- Data -- Inputs/Outputs
- Interaction between functions
- States

Data Example - Strings

- Empty string
- Very short
- Very long
- Length particular numbers (Lab 4)
- odd/even length
- NULL values
- Interesting characters in strings (with without null terminator, with without newline, with without special characters)

Interaction and States

- Functions
  - If A feeds into B, does B account for every return value of A?
- Loops and If/Else
  - Beginning and end of iteration (edges)
  - Variations in loop conditionals
  - Branch you’re not accounting for?

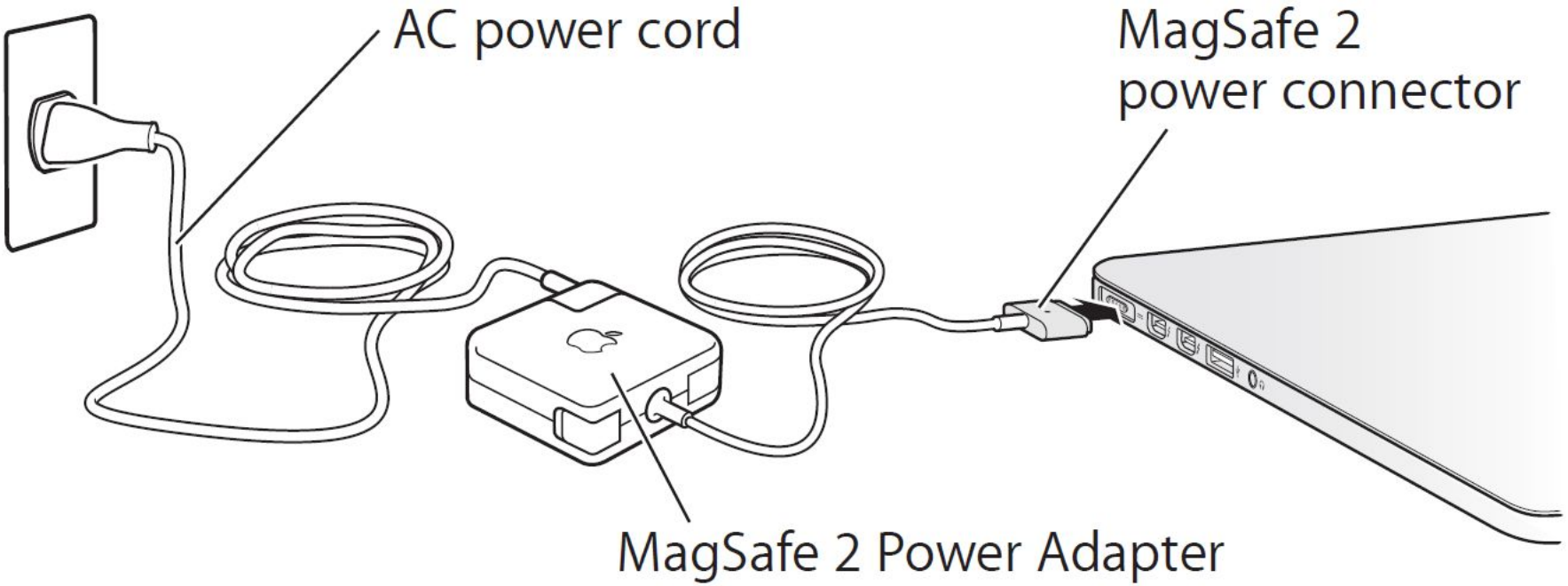
Interaction and States

- Dealing with system calls
  - Functions fail
  - Memory allocation failure
- User Input and Third Party (Client, Libraries/API)
  - Some covered in specs
  - Study specification/ ASK!

the art of debugging

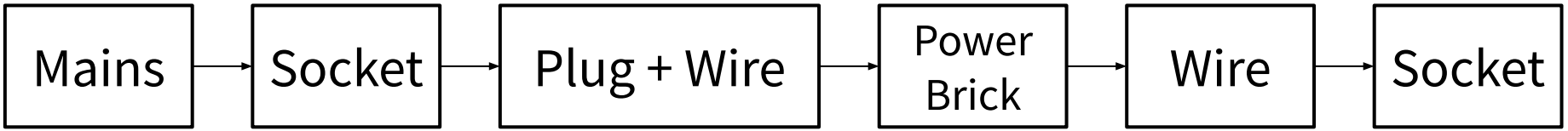
What is debugging?

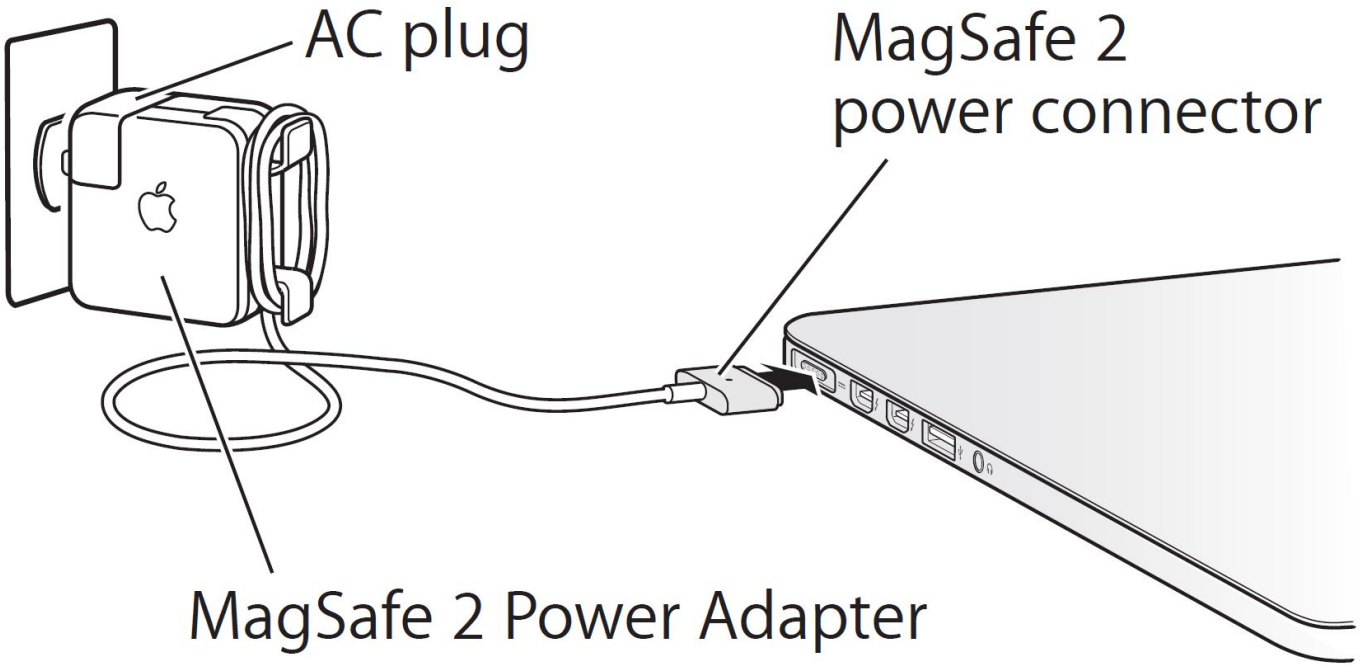
- (n.) the **process** of identifying and removing errors from computer hardware or software
- What is the process?
  - Need a generalized way of doing this

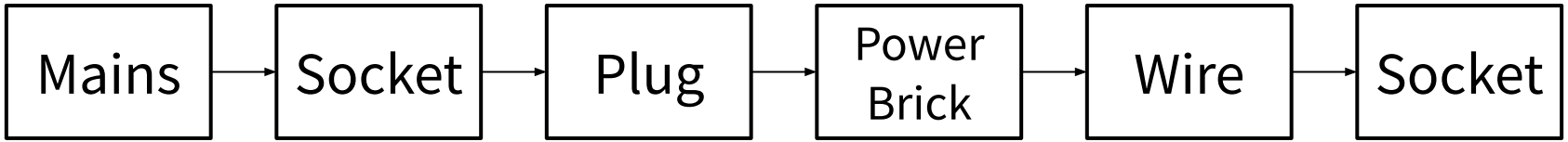

AC power cord
MagSafe 2
power connector
MagSafe 2 Power Adapter

What's the model?

- This charging setup is a system. What does that system look like?




AC plug
MagSafe 2
power connector
MagSafe 2 Power Adapter

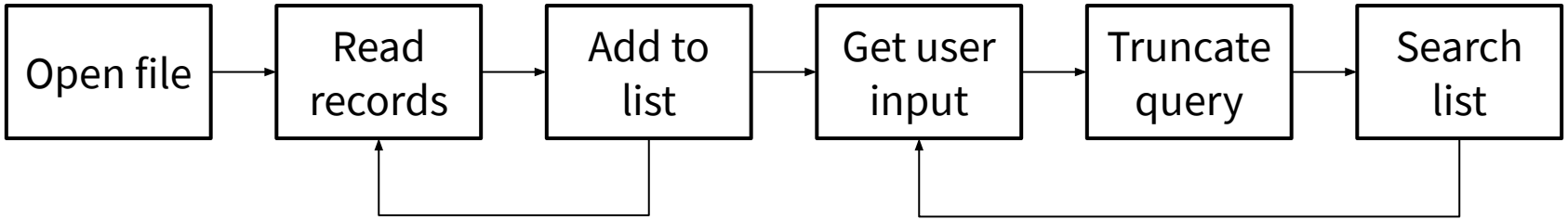

Mains
Socket
Plug
Power Brick
Wire
Socket

The Edwards Way to Debug

1. Identify undesired behavior
2. Construct linear model for desired behavior
3. Pick a point along model
4. Form desired behavior hypothesis for point
5. Test
6. Move point toward failure if point working, away otherwise
7. Repeat #4-#6 until bug is found

Exercise

- We’re writing mdb-lookup but lookups aren’t working. Where could the fault be?
  - What’s the undesired behavior?
  - What end-to-end components do we need to do a lookup?



the science of debugging

1. Choose example input

- Think about easiest to handle / base case input
- Manually evaluate expected output to memory level, if possible
- e.g. Lab 4, 5-char lookup string
  - input: “hi” → [‘h’, ‘i’, \n]
  - expected: [‘h’, ‘i’, \0]

2. Print value and compare

- Print and compare output values
- For strings, it might be worthwhile to check byte by byte to catch characters like \n, \r, \0
- If this matches your expected value, loop back to 1. and choose a more complex example or edge case example

3. Research + read

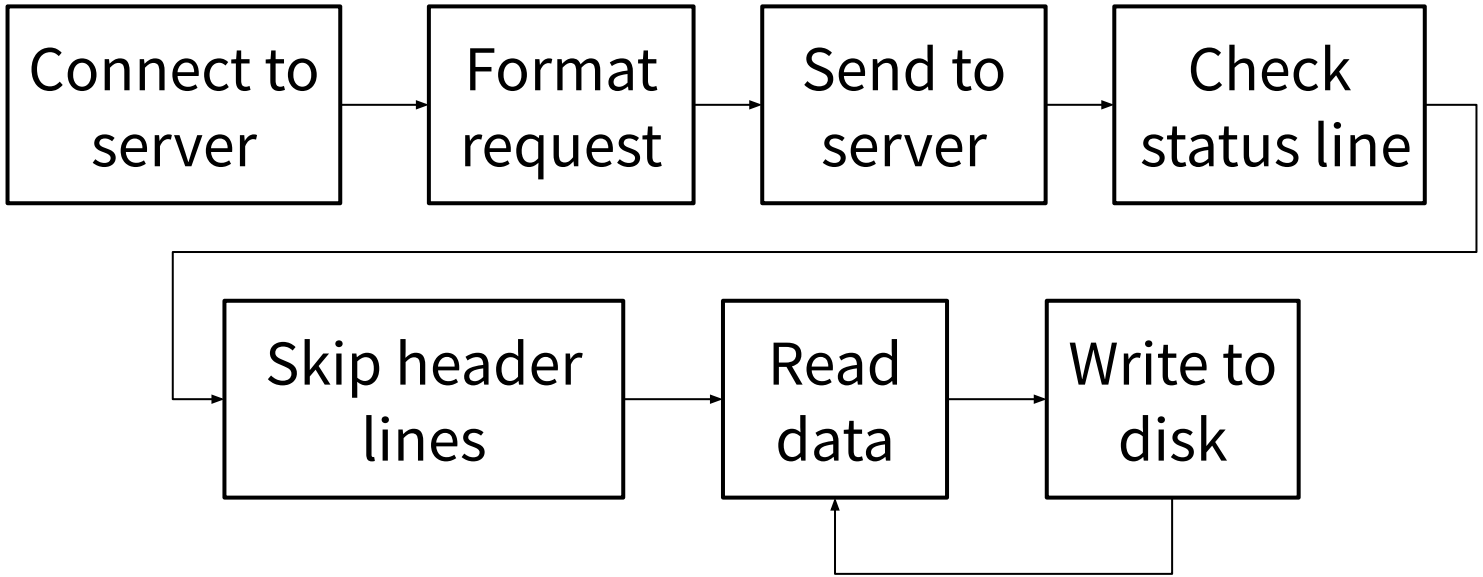
- Note differences in output value and expected value
- Review algorithm line by line with example input
  - draw diagrams if possible
- Check the man page
  - e.g. fgets reads in n-1 bytes

3. Research + read

- Check lab specs
  - e.g. Lab 6, HTTP \r\n instead of \n
  - e.g. Lab 6, not sending blank line at end of lookup
- Check similar issues on listserv
- Google error message, if possible
- Try not to go forward until example matches expected output
- Once resolved, you can loop back to 1 to try another example or move onto next node

(4. Hard-code and hold off)

- Try to avoid this step as errors trickle down and make further debugging harder
- If unable to resolve, temporarily hard code expected value to keep debugging further points of code
- Make note to come back and resolve initial error before submission


Connect to server
Format request
Send to server
Check status line
Skip header lines
Read data
Write to disk