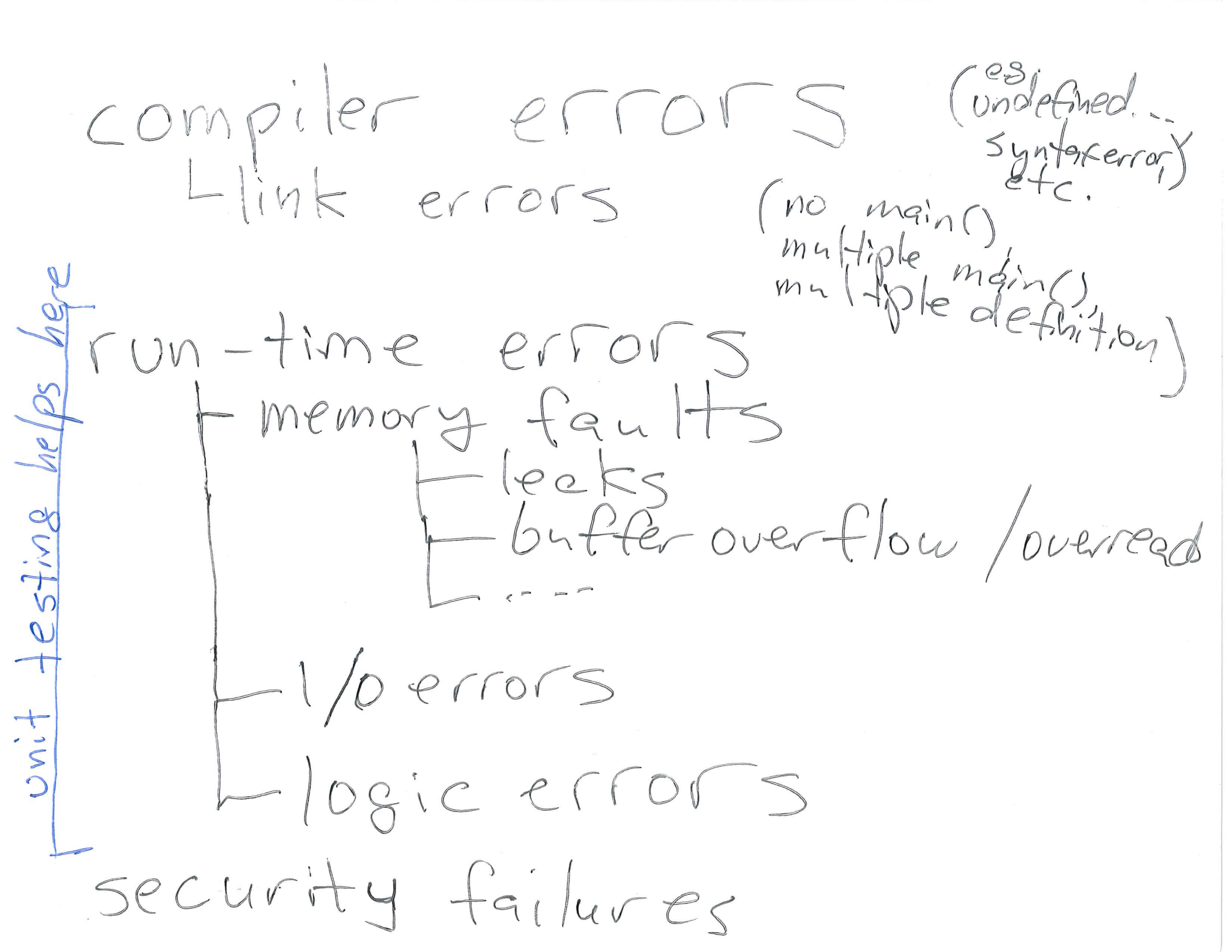
Objectives - Tue 4/2/2019

- Error handling
  - compiler errors vs. run-time errors vs. security failures
- Security: Buffer overflow attack
  - brief overview, in the context of reading BMP files
- How to think of test cases
  - easy cases, “edge” cases, “corner” cases, special cases
  - Think of how you can vary inputs (all sorts of inputs)
  - Regression testing - brief overview
- Exercise: Think of what to test in BMP file
- Vim tip: Open binary files without adding '\n'
- D.R.Y. Rule (refresh)
  - especially relevant this week for HW11 (BMP) test code

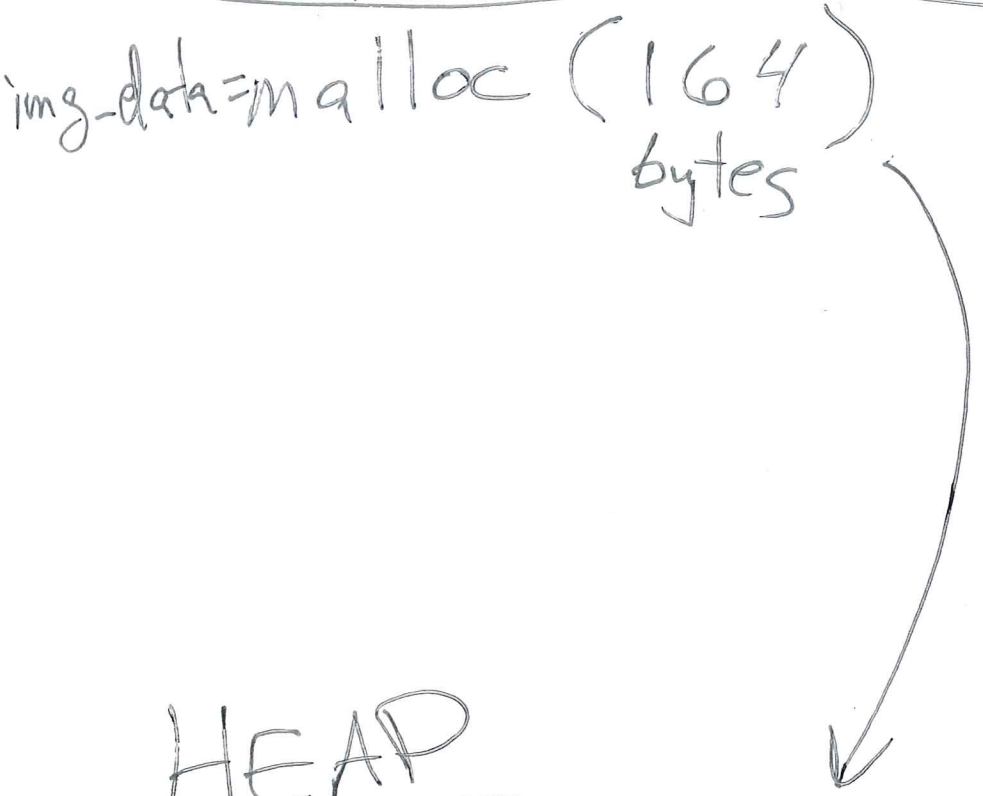

compiler errors
(eg. undefined... syntax error) etc.
link errors
(no main()
multiple main()
multiple definition)
run-time errors
memory faults
leaks
buffer overflow / overread
...
I/O errors
logic errors
security failures
unit testing helps here

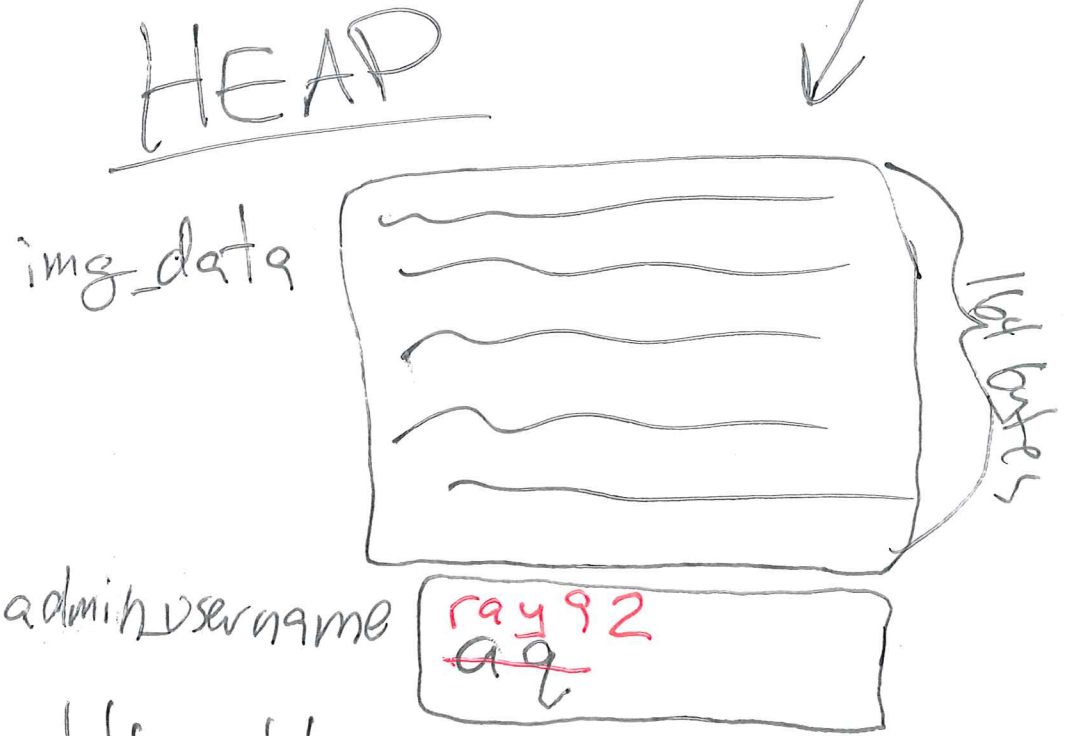
"Error"

- Compiler error
  - Compilation error (prob. 1 file)
    - Ex: Syntax error
  - Linker error
    - Combining files into an executable.
- Run-time error
  - Segmentation fault
    - Ex: read/write at NULL
  - I/O error
  - logic error
  - memory fault
    - leak
    - buffer overflow/overread
    - ...
- Security failure
  - ...

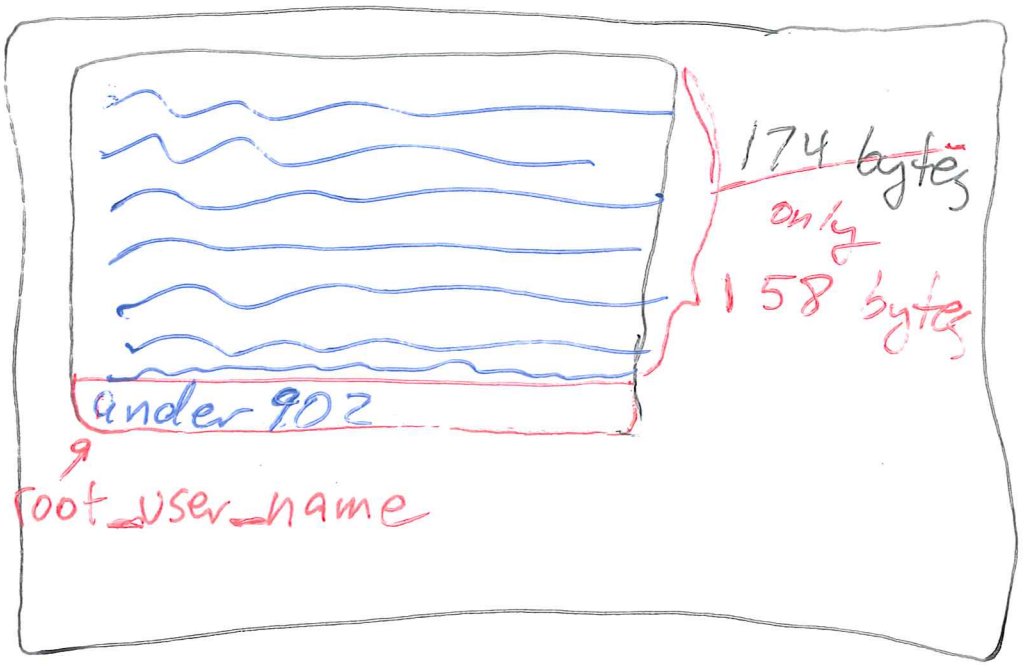
Detectable with unit tests

Buffer overflow due to invalid header data





Not same as HW 11


174 bytes
only
158 bytes
under 90Z
root_user_name

How can I think of test cases?

0. Put on your thinking cap.
   This is a creative process

1. Easy cases
   - Leverage your intuition
   - Easy to think of answer

2. Edge cases*
   Extreme values, sizes, magnitudes, etc. eg. INT_MIN

3. Corner cases
   Tipping point. Something changes. (Look for if(...) or while(...))
   e.g. print_integer(0) (9) (10)

4. Special cases
   "except when..." in specification

* Not universal terms

5. Vary external inputs or interaction

Regression testing

Retest everything.

Thoroughly

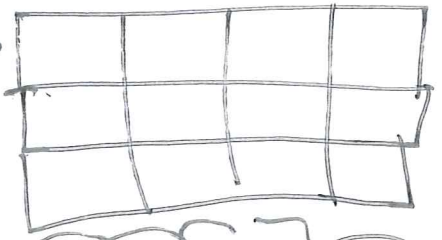
Systematically

Learning goal

Given a header / file / message format in the same spirit as BMP, 3, you should be able to think of possible errors and test for them.

NOT HW1

Hypothetically...



if 24 bit...
12 x 3 = 36 (12 bytes)

width = 4
height = 3
bits_per_pixel = 16

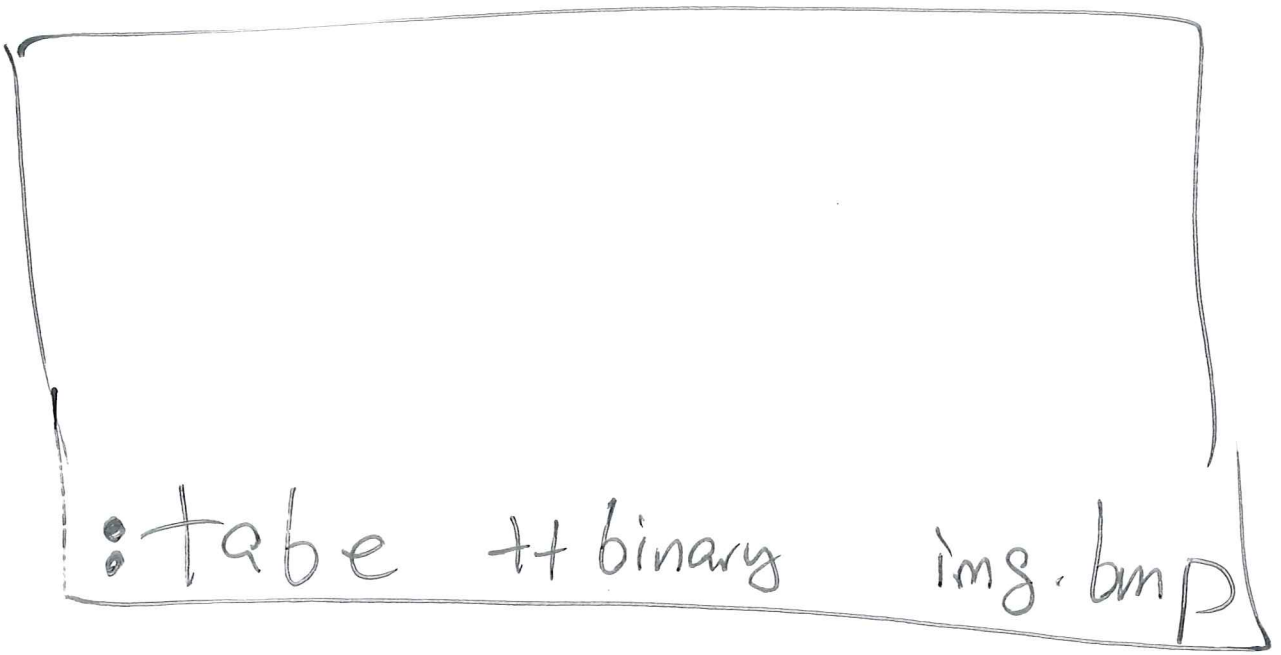
Every row's size (in bytes) must be an even multiple of 4.
Add 0x00 pad as needed.

image pixel data size in bytes.

width * height * bits_per_pixel / 8 + padding(...)

OPTIONAL

vim -b img.bmp



ax	78	120
cx	36	54
ax	qe	174

Right now:

Try to find as many error opportunities as you can. Work with someone near you to maximize your creative thinking.

Are there 10?

Look for redundant information in any form.

D.R.Y. rule

Don't repeat yourself

Copying (your) code for different conditions.
→ step back and refactor (e.g. helper)

Copying / rewriting a key constant value.
→ static const int BUF_SIZE = 128;

avoids compiler errors about multiple definition

Copying key logic (eg. equations, etc.)
→ Helper function

Representing underlying INFORMATION

Don't repeat error handling code
read_bmp(...)
check_bmp(...)