Debugging

Hadi Safari

University of Tehran

Advanced Programming
Spring 1398
(last update: July 25, 2019)

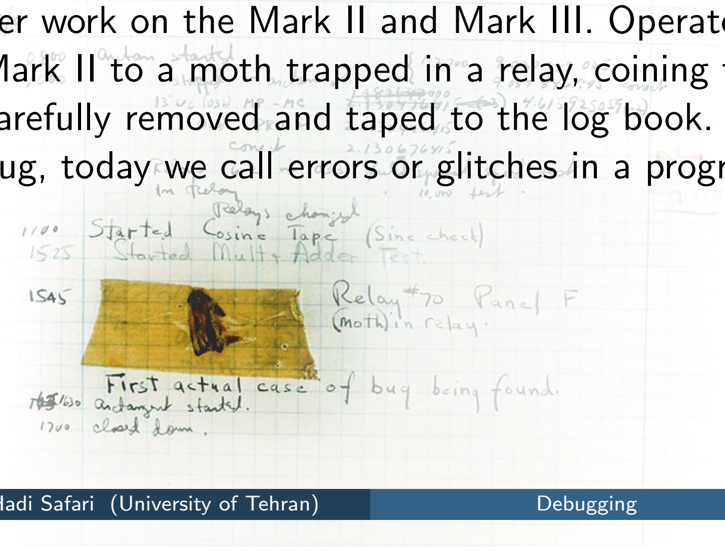
What & Why

What is *bug*?

- A **software bug** is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.
- a general word: fault $\longrightarrow$ error $\longrightarrow$ failure
- The process of fixing bugs is termed. . . **debugging**.

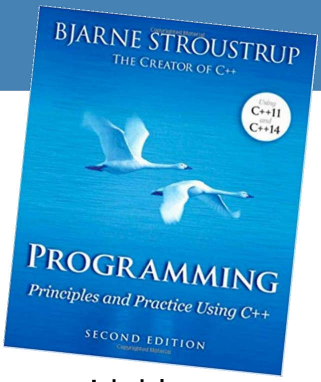
A bit of history

In 1946, when **[Grace] Hopper** was released from active duty, she joined the Harvard Faculty at the Computation Laboratory where she continued her work on the Mark II and Mark III. Operators traced an error in the Mark II to a moth trapped in a relay, coining the term bug. This bug was carefully removed and taped to the log book. Stemming from the first bug, today we call errors or glitches in a program a bug.



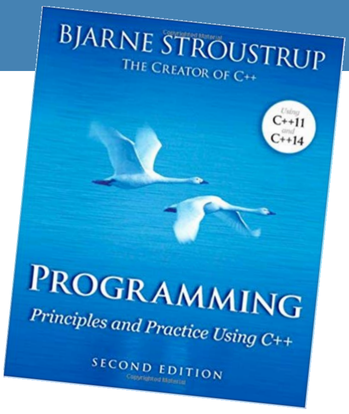
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.
1545 Relay #70 Panel F (moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

Does it happen?



- When we write programs, errors are natural and unavoidable.
- *The last bug* is a programmers' joke.
- By the time we might have, we are busy modifying the program for some new use.

Why does it happen?



- poor specification
- incomplete programs
- unexpected inputs & arguments
- unexpected state
- logical errors

Errors are always more common when you are tired or rushed.

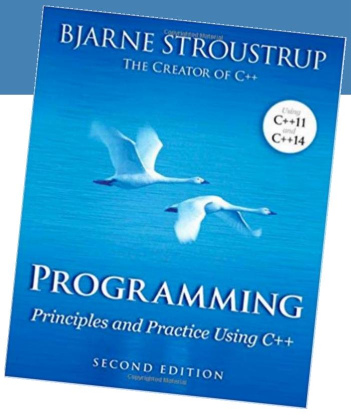
Does it matter?

- Therac-25 Radiation Therapy Machine $\longrightarrow$ overdosed six people
- Northeast Blackout of 2003 $\longrightarrow$ 55,000,000 people affected
- Pentium FDIV Bug $\longrightarrow$ \$475,000,000 cost
- NASA Mariner 1 Destruction $\longrightarrow$ \$18,500,000 cost
- Year 2000 Problem

What should we do?

- debug
- test
- formal verification
- design for test & debug, write clean codes

How far should we go?



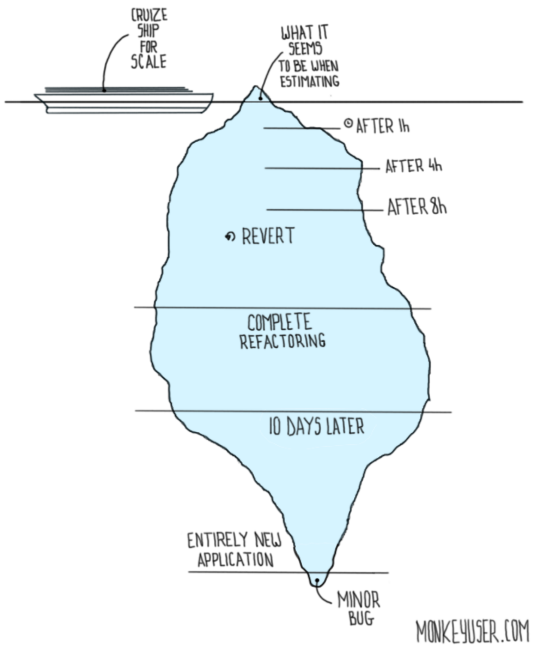
- Eliminating *all* errors?
- What about kicking out the power cord from the computer while it executed the program?
- What about data lose in safety-critical systems such as a medical monitoring program or the control program for a telephone switch?

Debugging is hard

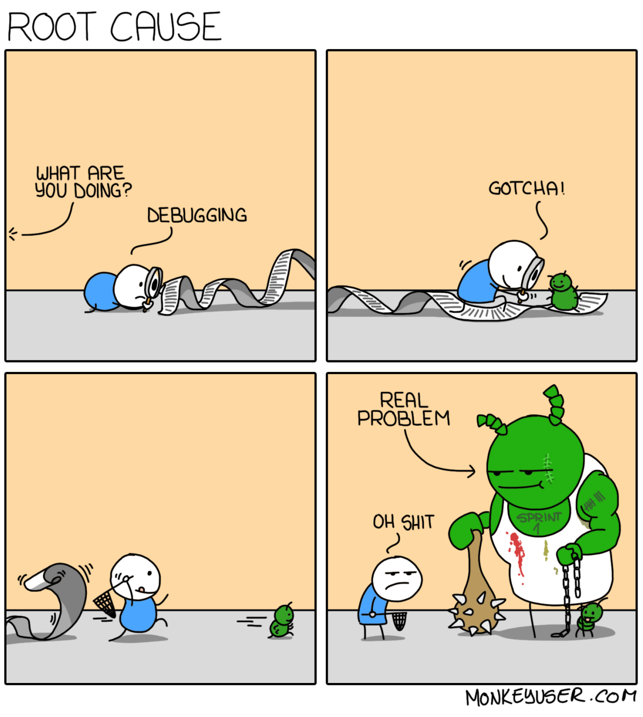
Everyone knows that debugging is twice as hard as writing a program in the first place.

— *Brian Kernighan*

Debugging is really hard

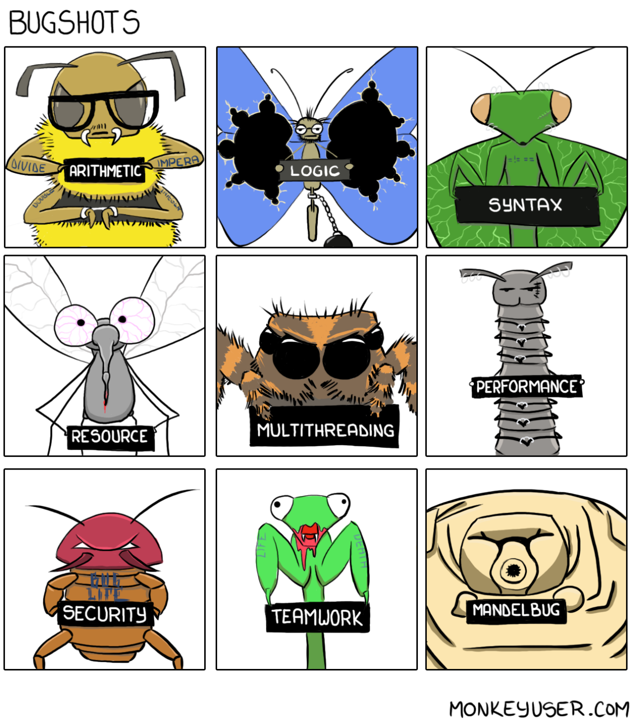


Bugs are so complex

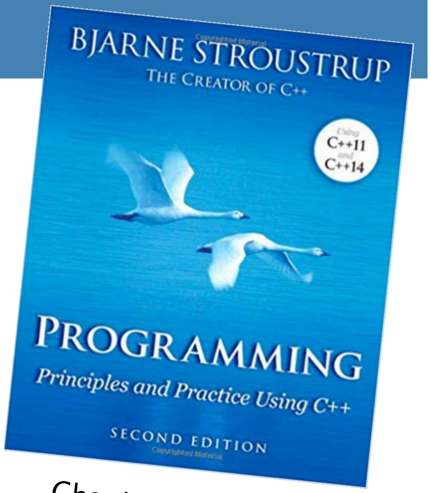


Types of bugs

Types of bugs



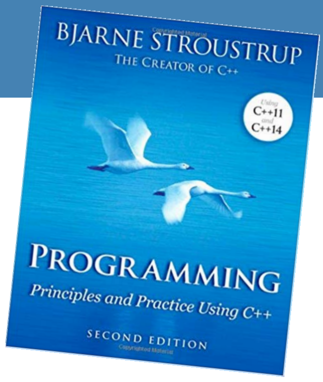
Types of bugs



Chapter 5: Errors

- Compile-time errors
- Run-time errors
- Logic errors

Compile-time errors

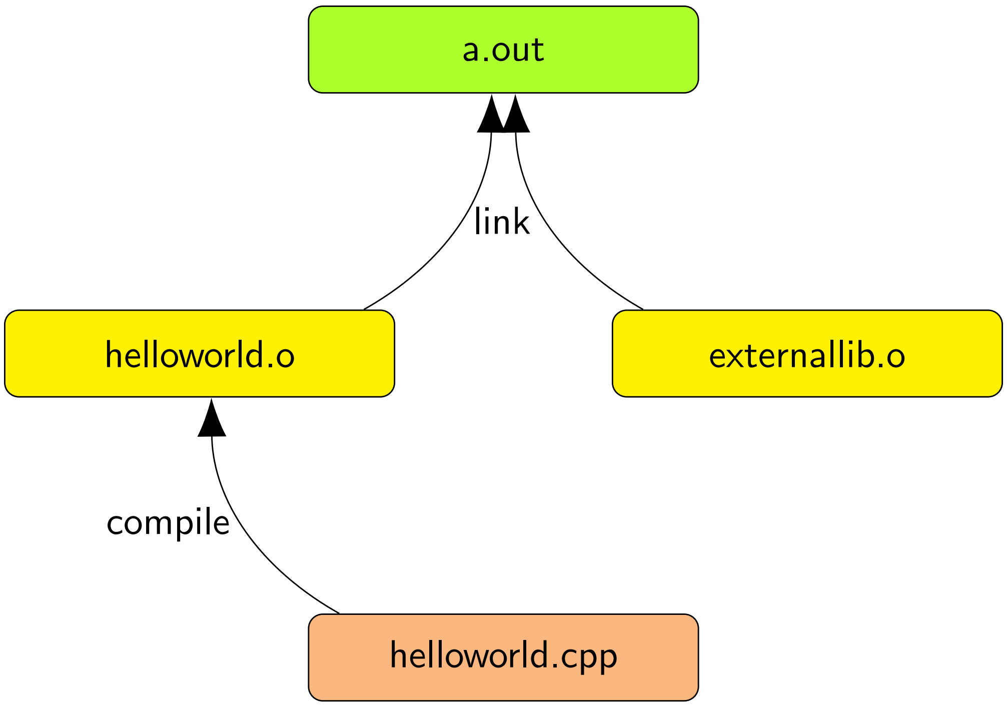


- compiler-time errors
  - syntax errors
  - type errors
  - non-errors $\longrightarrow$ As you gain experience, you'll begin to wish that the compiler would reject more code, rather than less.

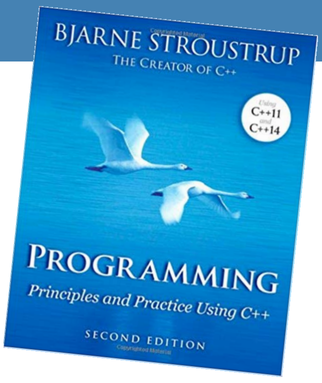
```
int area(int length, int width); // calculate area of a rectangle

int x = area(10, -7); // OK: but a rectangle with a negative width?
int y = area(10.7, 9.3); // OK: but calls area(10, 9)
char z = area(100, 9999); // OK: but truncates the result
```

- link-time errors

C++ build process

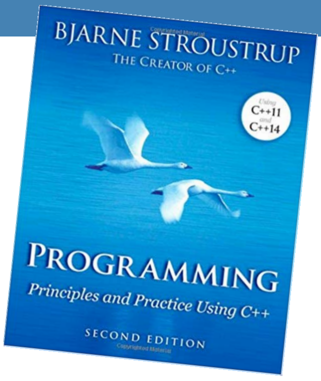

$$\begin{tikzcd}
 & \text{a.out} & \\
\text{helloworld.o} \arrow[ur, "\text{link}"] & & \text{externallib.o} \arrow[ul] \\
 & \text{helloworld.cpp} \arrow[ul, "\text{compile}"] &
\end{tikzcd}$$

Run-time errors



- run-time errors
  - integration errors
  - error reporting
  - range errors
  - bad inputs
  - narrowing errors
  - memory access errors

Integration errors



Who should deal with errors in function calls?

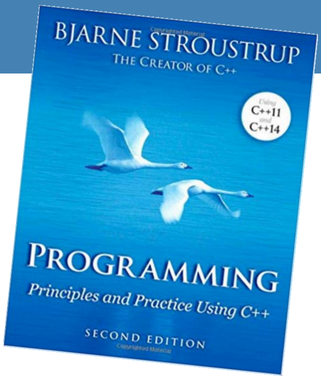
- caller
  - code duplication
  - are all calls error-checked?
- callee
  - we can't modify the function definition (e.g. library functions)
  - it doesn't know what to do in case of error
  - it doesn't know where it was called from
  - performance

So...
Check your arguments in a function unless you have a good reason not to.

Error reporting

- `cerr` & `stderr` ⟶ redirection: `./a.out 2> err.txt`
- return value
  - special values
  - `read`, `write`, `listen` (`-1`, in combination with `errno`)
  - C++ `main` function ⟶ `0`, `cstdlib` `EXIT_SUCCESS` & `EXIT_FAILURE`
  - `exit()` (`cstdlib`)
- flag
  - `errno` (`errno.h`), `perror` (`stdio.h`), `strerror` (`string.h`)
  - `stream` error state flags: `good()`, `eof()`, `fail()`, `bad()`
- exceptions
  - `throw` & `catch`
  - whoever could handle the error should catch the exception
  - rethrow: open files, dynamically allocated memory cells
  - `cstdexcept`
  - not to throw exception in destructors
  - inheritance & subtyping

Logic errors



- the most difficult to find and eliminate
- sources:
  - your understanding of the underlying program logic is flawed
  - you didn't write what you thought you wrote
  - you made some silly error
- estimation
  - Is this answer to this particular problem plausible?
  - How would I recognize a plausible result?

How to debug

False assumptions

The Art of Debugging, Ehsan Hajyasini, UT AP F96

Finding your bug is a process of confirming the many things you believe are true, until you find one which is not true.

- you believe that at a certain point in your source file, a certain **variable** has a certain **value**
- you believe that in a given `if-then-else` statement, the `else` **branch** is the one that is **executed**
- you believe that when you call a certain function, the **function receives** its **parameters** correctly

So. . . check the assumptions! $\longrightarrow$ binary search, pre & post conditions

More examples of false assumptions

What does debugging a program look like?, Julia Evans

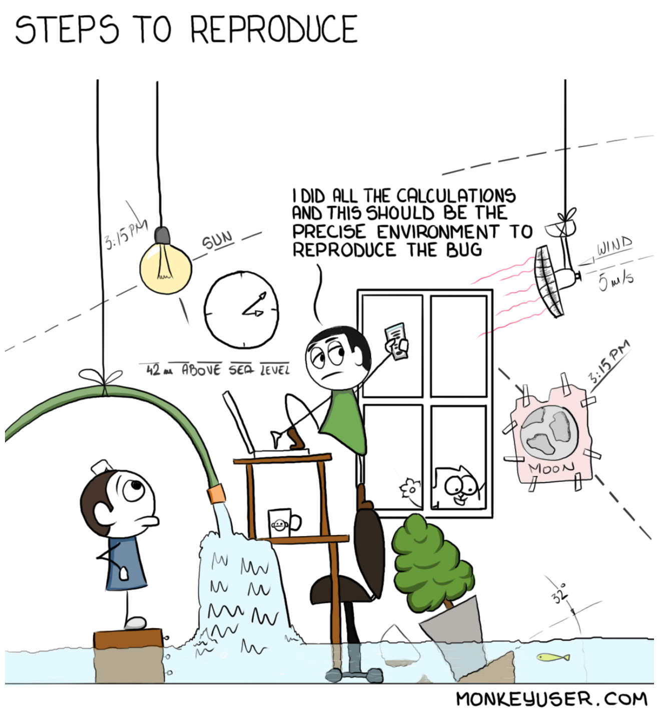
- this variable is set to X (“that filename is definitely right”)
- that variable’s value can’t possibly have changed between X and Y
- this code was doing the right thing before
- this function does X
- I’m editing the right file
- there can’t be any typos in that line I wrote it is just 1 line of code
- the documentation is correct
- the code I’m looking at is being executed at some point
- these two pieces of code execute sequentially and not in parallel
- the code does the same thing when compiled in debug / release mode (or with `-O2` and without, or. . .)
- the compiler is not buggy (though this is last on purpose, the compiler is only very rarely to blame :))

Stabilize, isolate, minimize

The Art of Debugging, Ehsan Hajyasini, UT AP F96

- make failure-inducing **input smaller** $\longrightarrow$ is more relevant, saves time
- make the program **crash faster**
- make the situation **deterministic** $\longrightarrow$ make bugs **reproducible**

Reproducing bugs

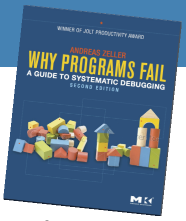

STEPS TO REPRODUCE
3:15 PM
SUN
I DID ALL THE CALCULATIONS AND THIS SHOULD BE THE PRECISE ENVIRONMENT TO REPRODUCE THE BUG
WIND
5 m/s
42 m ABOVE SEA LEVEL
3:15 PM
MOON
32°
MONKEYUSER.COM

Reproducing bugs

What does debugging a program look like?, Julia Evans

- for something that requires clicking on a bunch of things in a browser to reproduce, recording what you clicked on with *Selenium* and getting *Selenium* to replay the UI interactions
- writing a unit test that reproduces the bug
  bonus: you can add this to your test suite later if it makes sense
- writing a script or finding a command line incantation that does it

Scientific method of debugging


WINNER OF JOLT PRODUCTIVITY AWARD
ANDREAS ZELLER
WHY PROGRAMS FAIL
A GUIDE TO SYSTEMATIC DEBUGGING
SECOND EDITION
MK

Debugging, Max Goldman and Rob Miller, MIT Software Construction (6.031) F17

1. **study the data** $\longrightarrow$ incorrect results, failed assertions, stack traces
2. **hypothesize** $\longrightarrow$ where the bug might be, or where it cannot be
   - **slicing** $\longrightarrow$ When you have a failure the *slice* for that value consists of the lines of the program that helped compute the bad value.
   - **delta debugging** $\longrightarrow$ difference between successful execution and failing execution: test cases, diff debugging & undoing changes
   - **swap components** $\longrightarrow$ different implementations

   prioritizing hypotheses $\longrightarrow$ old, well-tested code vs recently-added code, library code vs your code
3. **experiment** $\longrightarrow$ devise and run an experiment
4. **repeat**

Stack trace

```
Traceback (most recent call last):
  File "./__main__.py", line 154, in <module>
    main()
  File "./__main__.py", line 145, in main
    config = extract_config(config_file_addr)
  File "./__main__.py", line 21, in extract_config
    config = DictWrapper(json.load(f))
  File "/usr/local/Cellar/python/.../3.7/lib/python3.7/json/__init__.py", line 296, in load
    parse_constant=parse_constant, object_pairs_hook=object_pairs_hook, **kw)
  File "/usr/local/Cellar/python/.../3.7/lib/python3.7/json/__init__.py", line 348, in loads
    return _default_decoder.decode(s)
  File "/usr/local/Cellar/python/.../3.7/lib/python3.7/json/decoder.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/usr/local/Cellar/python/.../3.7/lib/python3.7/json/decoder.py", line 353, in raw_decode
    obj, end = self.scan_once(s, idx)
json.decoder.JSONDecodeError: Expecting property name enclosed in double quotes: line 43 column 5 (char 1114)
```

Compiler logs

- compiler errors
  - compiler errors
  - link errors $\longrightarrow$ harder to read
- start from the first one
- not always at the exact position
- language & compiler version: incompatibility, better logs
- LLVM & `clang++`
- compiler warnings

Compiler flags

- warning options

`-Wall`	enable all the warnings about constructions that some users consider questionable
`-Wextra`	enable some extra warning flags that are not enabled by `-Wall`
`-pedantic`	issue all the warnings demanded by strict ISO C and ISO C++

- debugging options

`-g`	produce debugging information in the operating system's native format
`-ggdb`	produce debugging information for use by GDB

- sanitizers (`-fsanitize=`)

`address`	enable AddressSanitizer memory error detector
`leak`	enable LeakSanitizer memory leak detector
`undefined`	enable UndefinedBehaviorSanitizer undefined behaviour detector

External tools

`gdb`	the GNU Project debugger
`lldb`	LLVM debugger
`ddd`	a graphical front-end for command-line debuggers
`valgrind`	a programming tool for memory debugging, memory leak detection, and profiling