A Practical Introduction to Debugging

Jeremy Melvin

Sustainable Horizons Institute Webinar Series

8/24/2018

Introduction

- Research in STEM fields increasing rely on us to utilize computational resources
- Scripts - Python, Bash, Perl, etc...
    - Postprocessing/Preprocessing data
    - Managaing files/directories
    - Automating tasks
- Programs - Fortran, C, C++, etc...
    - Developing/Interfacing with libraries
    - Applications
    - Numerically solving mathematical equations

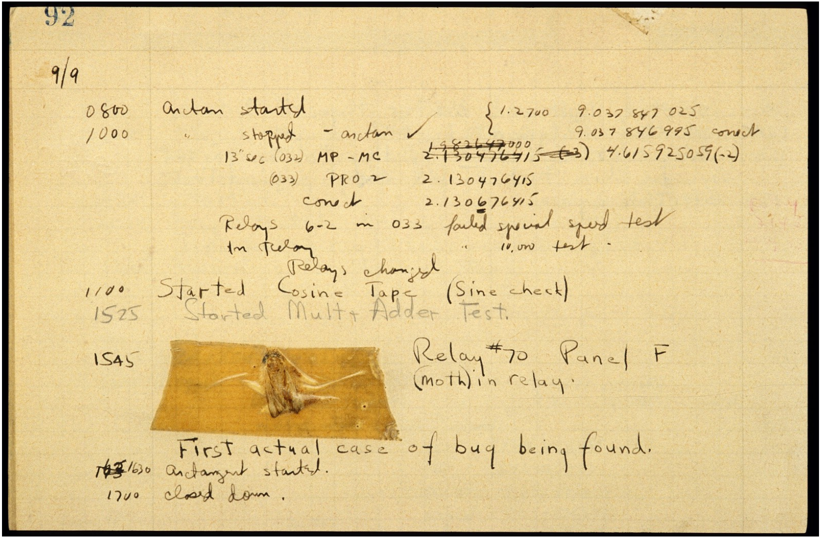
Introduction

- A possible outline for writing code:
  1. Develop/Learn theory (i.e. work with equations)
  2. Develop/Choose an algorithm to implement those equations
  3. Sketch out some psuedocode
  4. Implement in the programming language of choice

- Now, we're done right? In a perfect world, Yes!, but in reality, we are likely to make mistakes in implementation and thus we need another step:
  5. Make sure our written code is actually doing what it is suppossed to!

Debugging

- So we want our code to be “bug” free



92

9/9

0800 Antan started

1000 " stopped - antan ✓ {1.2700 9.037 847 025

9.037 846 995 conct

13" sec (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~ 4.615925059(-2)

(033) PRO 2 2.130476415

conct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10,000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult+ Adder Test.

1545 Relay #70 Panel F (moth) in relay.

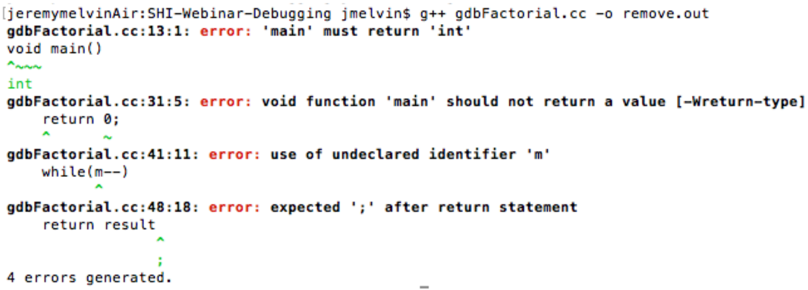
First actual case of bug being found.

~~163~~1630 Antangent started.

1700 closed down.

Types of "bugs"

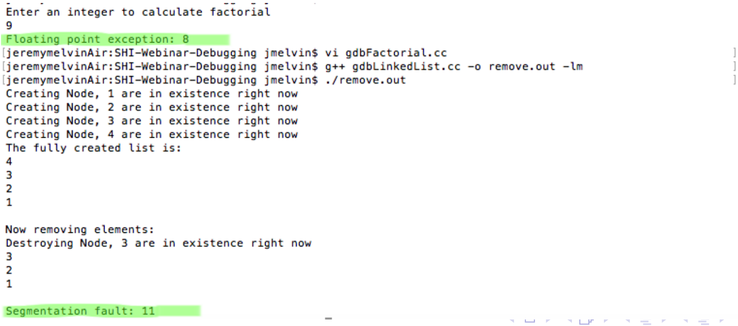
- Syntax Errors
  - Ex: Forgot a semicolon, Didn't declare a variable
  - Caught by the compiler, list of issues to resolve
  - Can't run your code until these are resolved
  - Usually, relatively easy to deal with



```
[jeremymelvinAir:SHI-Webinar-Debugging jmelvin$ g++ gdbFactorial.cc -o remove.out
gdbFactorial.cc:13:1: error: 'main' must return 'int'
void main()
^~~~
int
gdbFactorial.cc:31:5: error: void function 'main' should not return a value [-Wreturn-type]
    return 0;
    ^      ~
gdbFactorial.cc:41:11: error: use of undeclared identifier 'm'
    while(m--)
          ^
gdbFactorial.cc:48:18: error: expected ';' after return statement
    return result
                 ^
                 ;
4 errors generated.
```

Types of “bugs”

- Run-Time Errors
  - Ex: Segmentation Faults, NaNs, Floating Point Exceptions, Hanging
  - How can we approach these?
    - Print statements (Recompile/Rerun / Potentially lots of data)
    - Use a debugger (GDB / TotalView / Allinea DDT / IDE)



```
Enter an integer to calculate factorial
9
Floating point exception: 8
[jeremymelvinAir:SHI-Webinar-Debugging jmelvin$ vi gdbFactorial.cc                      ]
[jeremymelvinAir:SHI-Webinar-Debugging jmelvin$ g++ gdbLinkedList.cc -o remove.out -lm  ]
[jeremymelvinAir:SHI-Webinar-Debugging jmelvin$ ./remove.out                            ]
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Destroying Node, 3 are in existence right now
3
2
1

Segmentation fault: 11
```

Types of “bugs”

- Wrong Answers (helps if you know the right answer!)
  - Subtly Wrong
    - Very Difficult to resolve
    - Verification problems / Code Comparisons
  - Obviously Wrong
    - Find the wrong quantity and trace backwards to origin
    - Debugger can help with this process
- Memory Issues (Leaks, allocations, etc...)
  - Valgrind `http://valgrind.org/`
  - memcheck `http://valgrind.org/docs/manual/mc-manual.html`

GDB Introduction

- GDB (GNU Debugger) is a command line debugger (https://www.gnu.org/software/gdb/)
- Supports C, C++, Fortran and some others
- You may be able to use GDB with Python as well (https://wiki.python.org/moin/DebuggingWithGdb)
- Python has a built-in debugger called PDB which functions very similarly to GDB (https://docs.python.org/2/library/pdb.html)
- Other debuggers (DDT / Totalview / IDEs) typically have a more GUI based debugger but the basic commands and ideas we will discuss today should be applicable to all debuggers

Running with GDB

- **IMPORTANT: You need to compile with debug flags (-g or -ggdb)
- Launch with gdb: gdb –args* ./your_exe exe_runtime_args
- You can also attach gdb to an already running process
- See GDB Reference card for a partial list of GDB commands
  - Execution: run (r), continue (c), step (s), next (n)
  - Breakpoints: break (b), break if, clear, delete
  - Program Stack: backtrace (bt), frame
  - Display: print (p), display

*Should be two dashes - -, but doesn't seem to be coming through

Example 1: gdbFactorial

- Issue: Factorial calculation is returning 0 instead of the correct value
- Steps we will work through (commands in blue):
  1. **break** factorial (set breakpoint on factorial function)
  2. **run** (start program – will pause at breakpoint)
  3. **print** n (check to make sure our input is good)
  4. **display** result (keep an eye on what value result has)
  5. **display** n (keep an eye on the value of n)
  6. **next** (move through the function line by line)
  7. Fix: line 44: n → n+1

Example 2: gdbInterpData

- Issue: Last entry for interpolated data point is -nan
- Steps we will work through (commands in blue):
  1. **break** 43 (set breakpoint on line where nan is printed)
  2. **run** (start program – will pause at breakpoint)
  3. **clear** (clear the breakpoint on the current line)
  4. **break** if interp != interp (set breakpoint on nan)
  5. **continue** (resume execution of program)
  6. use **print** to track issue back to locIndR and locIndL
  7. **delete** (remove all breakpoints)
  8. **break** 33 if i == 10 (set breakpoint on LowerBound func)
  9. **step** (step into function... move frame reference)
  10. Fix: line 61: <= → <

Summary

- Introduction to GDB commands and how to approach debugging
- Just the beginning of what you can do with a debugger
- Many other things I use GDB for:
  - Learn a new code
  - Attach to an already running process
  - Use in parallel mpirun -np NP xterm -e gdb ./program
- I find GDB (or any debugger) a huge improvement to my efficiency
- If you have questions, feel free to email me any time: jmelvin@ices.utexas.edu