Debugging C programs

March 17, 2015

Consider the C program in `exponentiate.c`.
Its job is to calculate the approximation to

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}.$$

However, it is broken and return `inf` regardless of input.

Many examples stolen shamelessly from

`http://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html`

Debugging logical problems

If your code runs ok, but gives the wrong output in some/all cases, you probably have a problem with the logic. There are two main strategies for debugging logic problems

- Print statements
- Debuggers

Note that "staring at code" is not on the list!.

Debugging logical problems

Regardless of which of the two strategies you decide to use, you should pick an example case in which

- The problem is exhibited.
- It is easy to calculate by hand exactly what the state of the program should be at each line.
- The test case is as "small" as possible.

For example, in the code above, if $x = 2$ and $n = 2$ then the program should return

$$1 + 2 + \frac{2}{2} = 4.$$

Use `printf` statements to print out relative steps of the calculation, see what is going wrong.

- Example in `exponentiate_print.c`
- No new tools
- Adds code
- Sometimes you end up with a bug in the debugging code!
- Debugging code that can be turned on and off with a flag is nice.

Using GDB

If we include the compile flag `-g` then we may use `gdb` to debug our program.

- `gdb a.out`
- Add a breakpoint with `break linenum` or `break function`
- Run with `run`
- Step into next line with `step`
- Proceed to next line with `next`
- Print variable with `print`
- Repeat last command with return
- Quit with `quit`

See
`http://cs.baylor.edu/~donahoo/tools/gdb/tutorial.html`

Using valgrind

Valgrind is a SUPER useful tool for finding problems with memory access:

- Reading memory you shouldn't
- Writing memory you shouldn't
- Not freeing memory
- Using uninitialized variables