Debugging Haskell Programs

Advanced Functional Programming
Spring 2014

Kinds of debugging

- Finding and fixing typing errors
- Finding and fixing errors in logic
- Finding and fixing infinite loops
- Making programs run faster
- Finding and fixing space leaks

Tools for debugging

- print
  - Requires that your program be in the IO monad
  - If its is in some other monad, you can add the OutputT transformer.
- Debug.Trace
- Using the GHC stack trace mechanisms
- Using the GHC profiler
- The GHCi debugger
- Other external libraries for debugging
  - The Hood debugger
  - The HAT debugger (The Haskell Tracer)
  - The Safe library

Preventative Debugging

- Good testing tools can help you identify bugs before they become show stoppers

- Hunit
- QuickCheck
- SmallCheck

Finding typing errors

- Comment out some of your program.
  - Add stubs for the commented out part

```
f:: A -> B
f x = undefined
```

- Force the part whose type you can't figure out into a context whose type is known.

```
known:: Int -> a
known n = error "call to known"
… (known x) …
```

  - Will raise an error that tells you the type of x isn't Int, but at least you'll know what type it is!

Finding infinite loops

- Use the profiler. Unlike some tools it still creates a xxx.prof file even if xxx is interrupted by ^C
- Then look at what function has the most (or suspiciously many) calls.

Inspecting inside infinite loops

- Use print (if your in the IO monad)
- Use Debug.trace
- Add an extra argument to the function.
  - Decrement this counter for every recursive call.
  - Add a case like
    - `f count xs | count <0 = error (…)`
- Use the GHCI debugger (I can never understand what's going on if I do this)