

Object Lifetime and Pointers

CSCI 400

Colorado School of Mines

24 August 2017

Why do we care?

Could affect:

- Performance
- Reliability
 - e.g. Ease of debugging
- Language choice

Object Lifetime

- **Lifetime of a variable**
 - Time during which the variable is bound to a particular memory cell
- Ruby built-in objects created when value assigned
 - e.g. `x = 5`
 - Other classes create with `new`
- Factory methods also create objects
- Ruby uses *garbage collection*
 - Destroys objects that are no longer reachable

Object Lifetimes

- 1 Static
- 2 Stack dynamic
- 3 Explicit heap
- 4 Implicity heap

Variables by Lifetime: (1) Static

Static

- Bound to memory cells *before execution begins*
- Remains bound to same memory *throughout execution*
- All FORTRAN 77 variables, C `static` variables
 - But *not* C++ class variables

Variables by Lifetime: (1) Static

- *Advantages*
 - Efficiency – direct addressing
 - History-sensitive subprogram support (TODO: what?)
- *Disadvantages*
 - Lack of flexibility (no recursion TODO???)
 - Storage can't be shared among subprograms (TODOagain: what?)

Variables by Lifetime: (1) Static

TODO: 'Quick Ex'?

Where are static vars stored?

(Assuming C/C++)