

Offset	Topic
00:17	<ul style="list-style-type: none"> • Intro
	<ul style="list-style-type: none"> • Terry Pratchett diagnosed with Alzheimer's
03:49	<ul style="list-style-type: none"> • Word of the Week: crash
	<ul style="list-style-type: none"> • http://catb.org/jargon/html/C/crash.html
05:18	<ul style="list-style-type: none"> • Inner Chapter: Programming Paradigms 2
	<ul style="list-style-type: none"> • Data structures <ul style="list-style-type: none"> • Refresh on variables, storing simple values • Simple being typically some form of number • Incremental step between simple values, data structures is an array • A series, usually a fixed number of them, of simple values, all the same type • Data structure is a mixed bag of simple values • Each property or element in a structure has a name or index • Scopes, local and global <ul style="list-style-type: none"> • Local is defined in a procedure • Has to be passed as a parameter value • Can get cumbersome • Some languages allow global variables • All code every has access to the same data • Can be dangerous, don't know as cleanly how that data changes • Managing state <ul style="list-style-type: none"> • Without data structures, you have a set of unrelated simple values • Collecting them together communicates intent, part of the same thing • Initially refer to state as an oblique reference to a state machine • http://en.wikipedia.org/wiki/State_machine • Procedures that change structures create an emergent state machine • Understanding transitions, interactions is helpful • Helps bound partitions of unrelated state, unintended effects • Dynamic structures <ul style="list-style-type: none"> • Fixed sized structures are only so useful • Need to allow for however much state is needed ahead of time • Can run into bounds problems or waste memory • A data is just an address in memory, information about the byte layout in memory • What if you could grab a chunk of memory at runtime, declare its structure? • This is dynamic programming • Usually involves a pointer

Offset

Topic

- Pointer points at memory, carries type which says how big, what structure
- Compare array to linked list
- Fixed structures can be more efficient, know read sizes ahead of time
- Dynamic can be less efficient, may have to read more, traversing pointers
- Increases complexity of state management
- What if some procedures only make sense to some structures?
 - Can do this by convention, use modules to associate
 - What is some of the state is changes as a side effect of other changes?
 - Can't do that with pure procedural programming
 - All parts of a structure are equally accessible to all procedures
 - OO combines structures and procedures into the concept of an object
 - Procedures on objects are usually called methods
 - Structures on objects are usually called fields, attributes or members
 - Other code sends messages to objects via those methods
 - Can simply be thought of as calling an object's method
 - The distinction is important to some languages, like Small Talk
 - Means that some part of an object state can be hidden
 - Methods can change but not external callers
 - This is encapsulation, hiding implementation to any code outside of an object's definition
 - Object definition is typically called a class
- Inheritance, polymorphism
 - These are other qualities often ascribed to OO
 - I think this is more specific to some early OO rather than being critical to OO
 - Inheritance means that there can be a hierarchy of types
 - Polymorphism means that a caller can call a method on a parent or super type
 - Object of child type can have its own, different behavior
 - Getting this right is problematic
 - Anything that depends on aspects of state visible outside class are easily broken
 - Commutation of equivalence comparison
 - Point, color point example
 - Interface inheritance is typically safer but often dismissed
 - Too many consider implementation inheritance as primary means of code reuse
 - Composing objects is safer method of reuse most of the time

Offset

Topic

- Languages with more sophisticated access models, like C++ and friendly, make composition better
- Start to think about responsibilities and collaborations
- Especially with dynamism, procedural code can do all OO can
 - C++, one of the most popular OO languages was original built with C
 - Pointers to functions made it possible to associate behavior in structures
 - OO can be done with pure functional classes and pure data structures
 - The questions that are more important are about the state being managed
 - What parts of the state should be public?
 - What parts should be private?
 - All of the public parts combine to define a combinatorial state space
 - Private do too but can control the subspace that is reach
 - Have to be careful as sometimes public, private is fuzzy; side effects
 - No private state, don't need OO strictly speaking
 - Need to make assurances about side effects, related field changes, need OO
 - Can do by convention but opens the door for coincidental effects, other mistakes

32:58

• Outro

- Contact me
 - Email to feedback@thecommandline.net
 - Web site at <http://thecommandline.net/>
 - IM to command.line@skype
 - Listener comment line is 240-949-2638
 - del.icio.us tag is "for:cmdln"
 - <http://twitter.com/cmdln>
- I'd like to thank libsyn.com for AAC hosting and Wouter de Bie for MP3 hosting
- These notes and the show audio and music are covered by a Creative Commons license
 - <http://creativecommons.org/licenses/by-nc-sa/3.0/us/>
 - Attribution, non-commercial, share alike