Composite types
Ch 8 Scott

The line between “built-in and composite types is thin
Is a string built in?
   Not in C
What defines a composite type?

Record / structs
   Go - struct
   Kotlin — any class, but especially data classes (Data classes are really supposed to have no code)
Why have records?
Implications of reference model vs value model on records
Is Go anonymous include equivalent to inheritance in Java/Kotlin??
copy
   a <= b
   what is difference in Go and Kotlin
   again value-model vs reference model language
   copy and copy constructors in Kotlin. (clone in Java — big debates over it)

see equal_go and copy_kt
   in particular, for go show the addresses of objects in equal_go

Arrays
   usually homogenous type
   Why homogenous????
   value-model language it is kind of required
Java / Kotlin since everything inherits for Object can make non-homo array
easy in reference model language
   Note that similar game is harder in value model Go
   usually contiguous in memory
Go — arrays MUST be sized at compile time!! (Why?)
   arrays contain the objects, literally. So each spot in otherwise “empty” array actually contains the sting with zero value(s).
Kotlin — Arrays contain references (what a shock)
Go — slices contain REFERENCES!!! Why? SO?
   consider difference between
   a := b for array and slice in Go
   for array, everything is new! Copying can be expensive
   for slice, the address of the slice is new (value model)
   but all the content is the SAME (copy the references)
   WHY?
Subsections of arrays
   go slice[start:end] returns that part of slice between start and end
   kotlin — array.slice(start..end)
   slice takes array and returns a list!!! Why not array?
   Also list.sublist(start..end)
   slice is a new object, sublist pointer within existing list
Java: neither arrays not ArrayList have subsections built in.
Heap allocation vs stack allocation!!!

Row-Major & Column major ordering
assumes array contained in contiguous block of memory
Looking at pointer addresses in Go you can see this.
Suppose A is 7x10 array
R-M
C-M
  a[2,4], a[3,4] ... a[9,4],a[0,5]
Why do I care?
  Max performance says always access memory locations near each other
  so nested for loop for R-M
    for i 0..6
    for j 0..9
      a[i][j]
  For C-M
    for j 0..9
    for i 0..6
      a[i][j]
Easy to build multi-d array in RM

Composite equality checks
Go == on structs compares the stuff inside — a deep check. (again, kind of natural in value model)
  Go defines == over array and does a deep check!!!
  no == over slices!!! Why? (slices could contain themselves, Why is this a problem?)
    see equal_kt, equal_go

Associative arrays (maps), sparse arrays, ...
  are these really arrays? Or are they something else that just uses the same syntax?

Strings:
  are they a primitive type in the language
  C — definitely not
  Java, Go, Kotlin — might as well be.
  J,K,G — String is a fixed entity. A length change (append) makes new string
    Java StringBuffer, StringBuilder
  Go: “A string is an immutable sequence of bytes”

Recursive types
  E.g. Linked lists
    In reference model languages these are natural
    How to Handle in Value-modellangs like Go.
Answer Pointers!!!
  see pointer_go

  see tree_go — lots of points to make
new operator in Go / Java allocates from heap. stack allocation auto reclaimed when frame complete (closures aside), but heap is forever!

Garbage collection

Reference Counting
when the number of references goes to zero, reclaim
problem — circular structures
problem, how to count
fragmentation of memory

Mark-and-sweep
1. mark everything as useless
2. start with all non-heap pointers and recursively follow. Mark everything touches as good
3. Go through heap and destroy everything not marked as good

Stop and Copy
split memory in half
Rather than mark and sweep, in step 2, copy from current to new. Then delete anything not copied. Next time, switch current and new

Lists, etc
difference between list and array?
pointer following?
typically not indexed, but lists in Kotlin are
Go: no list type?
Homogeneous vs heterogenous
Opinion: lists are associated with functional programming because they are one with LISP. Otherwise, lists are fairly boring.
LISP and the ability of LISP to write/run code. Maybe a little in Emacs? (or not)