Composite types
Ch 8 Scott

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

Record / structs
    Go - struct
    Elixir defstruct within defmodule
    Why have records?
    Implications of reference model vs value model on records

    Is Go anonymous include equivalent to inheritance in Java??
    What is stored in a go struct?? Overhead??
        see size_go/structsize.go

    copy and Equality
        a==b
        what is difference in Go and Elixir?
            again value-model vs reference model language
            see **equal_go/equal.go**
                in particular, for go show the addresses of objects in equal_go

        Question: is elixir value model or reference model?
            A: given immutability it really does not matter — Why??
        Equality in Elixir:
            seems to be a deep comparison. see **equal_ex/equal.ex** But it is hard to be certain
        Copy Elixir:
            probably just a reference — again immutability makes it hard to tell and renders the discussion somewhat irrelevant
            There is no way to see pointers in Elixir

```
iex(1)> a=[1,2,3]
[1, 2, 3]
iex(2)> b=[0|a]
[0, 1, 2, 3]
iex(3)> c=a++[4]
[1, 2, 3, 4]
iex(4)> a
[1, 2, 3]
iex(5)> b
[0, 1, 2, 3]
iex(6)> c
[1, 2, 3, 4]
```

Question is the [1,2,3] of a used in b or c??
    Almost certainly but immutability means it does not matter

Arrays

- usually homogenous type
    - Why homogenous????
        - value-model language it is kind of required
            - Go array vs Slice what is stored where
  - Exactly What is stored in an array in Java
  - Java since everything inherits for Object can make non-homo array
    - easy in reference model language
      - easy with subtype polymorphism
    - 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).
- Elixir — no arrays — why not
  - are tuples in elixir a substitute for arrays (they are indexed)

```
iex(1)> aa = {"q", "w", "e", "r", "t"}
{"q", "w", "e", "r", "t"}
iex(2)> elem(aa, 2)
"e"
```

- Go — slices contain REFERENCES!!!   Why? So what?
  - 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?

    - 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
  - A[2,4] followed by A[2,5] … a[2,6],a[3,0]
- 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 so almost all languages use Column-major

see size_go/sizeof.go

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?)

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, Elixir — might as well be.
    - J,E,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”
- Why are strings immutable????
  - String Pool
    - a place to store string literals
    - String pool — I imagine as a hashtable<String, String>
    - In big apps string pool can save lots of space
      - see pool_java/Pool.java
  - Security
    - anti hacking. Mutable strings could let hackers attack. For instance, person passes a string — we validate — in background they change ….
  - Thread Safety
    - immutable strings are thread safe
- Note that all of these arguments in favor of immutable strings can be generalized to immutable everything!

Recursive types

- E.g. Linked lists
  - How to Handle in Value-model langs like Go.
- Answer Pointers!!!
  - see **pointer_go —** already discussed so this code is review
  - 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 (why not??)
- Go: no list type?
    - as a package, but NOT a language primitive
- Homogeneous vs heterogenous
- Opinion: lists are associated with functional programming because they are one with LISP.

    Counter argument. A: Lists can be built recursively by appending to the front. In so doing you can add items to list without changing the list as it was previously seen. Lists built in such a way are therefore perfect fit for functional programming.

    B: Linked lists are amenable to immutability — indeed immutability makes sharing of linked list parts a practical thing
- For beginning of an implementation
    - Elixir: LL_ex/LL.ex
    - Go: tree_go (a tree rather than a linked list

- Subsections of arrays/lists
    - go slice[start:end] returns that part of slice between start and end
    - Java: neither arrays nor ArrayList have subsections built in.
    - Elixir: Enum.slice gives subsection of linked list.