Topic 5 — Control Flow Ch 6 Scott
sequencing
selection
iteration
procedures
recursion
concurrency
exception handling
nondeterminacy (skip)

"sequencing is central to imperative languages but plays a relatively minor role in functional"
We will see that sequencing in Elixir can be fairly important in that the order in which functions are defined in a module affects the matching process.
Question, what order are args to a function evaluated?
Go: left to right at least usually. see order_go
Book notes that optimizing compilers might want to change order. So, even if you test — like order_go — probably best to not rely.
in infix notation, what order are operators evaluated (note no precedence problems in pre/postfix)
C: On the order of 15 levels of precedence — about that many in Java and Go
also need to know "associativity"
10-5-5 is this 10 or 0??
17+MAX_INT-50
if r->I then MAX_INT-33 else
In a program I wrote with math ALL I had was one level of precedence and L-R associativity
Good idea?? Smalltalk, APL do it

"side effect" if expression evaluation influences subsequent computation in any way other than by returning a value
NOTE—by this definition printing is not a side effect
if there are no side effects then a lot of sequencing is not important.
Explain!!!!
ALL assignment statements are done for side effect. If you did not care about the value being stored, do not store it.
purely functional languages have NO side effects.
how do you write a program that has no assignments??
lots of function calls — or at least make a new scope.

When using a "Value model of variables" — GO/Java primitive
a = b+c
l-value — the location of the variable. Where a thing is to be stored. (a) Think of L as standing for both Left and Location
usually l-value is simple thing but it can get complex due to arrays a[b[5]+3] and structs/objects
r-value — an actual value (b,b,b+c)

reference model of variables
still have l-value and r-values
every variable is an l value so when an r-value is needed need to "dereference" — that is turn it into an r-value.

suppose you did not know if Go / Java were value model or reference?
Design an experiment to determine
What gets in the way of experiment?
Can you tell value or reference on an immutable variable
Does Go have immutables?
Strings are immutable (how can you tell)?
Otherwise NOT (except for things declared const)

Go almost always uses value model — exception data structures like slices
A slice is a reference. a sub slice is a reference to a location within a slice!!
Question: what information is actually stored about a slice and why?
4 things: starting location in memory
type of information being stored
how many pieces of information
how much space you have to store things
see PoiSli_go
Why does slice need to know all of this???

Value model languages DO NOT have aliases (by default). They can't. But even in value-model langs, references and aliases can be really handy. So they have pointers! Reference-model does not need because everything is a pointer.
Go: need to tell that you have a pointer when passing into funcs, but after that value and pointer are treated the same — from programmers perspective.
see pointer0_go — using pointer to get an alias
pointer_go — using pointers in function calls
nullpointer_go — even pointers in Go have a "zero value"

Initialization and the problem of uninittialized vars
Note that value and reference model langs have different issues
Java: every value model starts as 0. Every reference starts as null
Go: every type has a defined "zero" state. Every var initialized to zero state.
pointers?
Java — definite assignment guarantees that no variable is uninitialized. Unneeded in Go. Is it really needed in Java?
see Definite.java in short_go
Short circuit boolean
see short_go
Note unlike Java Go does not allow assignment with boolean
also a = (b=6)

Flow

break — a limited for of goto??
allowed in both Java and Go.
see break_go

Labeled loops and break:
break allow you to get out of loop early
labeled loop allows you to not just get out of inner loop
GO: break cannot get out of current function — WHY?

break_go

return — should it be allow from anywhere or only at end!
should I be able to return from more than one func.
very rare
mullti-level return — why not?
e.g. return3 — causes return,return,return so pop function stack 3 times with one statement.
Crossover between exception and **multilevel return**. Note that can simulate a multilevel return in Java using exceptions — Write Example

---

SEQUENCING
"sequencing is central to imperative programming"
because imperative programming makes heavy use of assignment statements

SELECTION
the if statement
the switch statement
switch_go and switch.go, Switch.java therein
"the principle motivation is to facilitate the generation of efficient target code"
switch in Go
NO default fall through
fallthrough statement
but allows listing of multiple cases
any type that allows an == comparison
tagless optional

LOOPS
another imperative concept

Iterators and enumeration controlled loops
rather than just using numbers allows programmer to do a loop for everything in a collection
Have seen this in GO slic_go/slic.go
for idx,val := range slice {}
Java: looping over array or any collection
it is really clunky (to write) need to create another class and implement n interface (effectively creating an object closure)
see readCSV_java

Using closures to simulate enumerated loop
on homework 3
Any number of syntaxes for for loops

Recursion
Advantage: no special syntax (but does require support for recursion)

Why NOT have recursion support?
without recursion, every function can exist in a preallocated space so stack operations minimal — only thing you need in the stack is the resumption point
So why have recursion?
Some problems are naturally recursive
towers of Hanoi
Merge sort
QuickSort
These can be implemented with iteration and a stack, but recursion is neater.

Tail recursion:
see tailrec_go
“additional calculation never follows recursive call”
In this case you can do the recursion without adding to the stack. Just use/overwrite the stack. Since new stack frame is one of the principle costs of recursion…
NOTES WITH tailrec_go. GO does NOT have TR optimization . Java does not have it. When you run this program, it kind of looks like tail call optimization is there. But this is a fiction. How to tell it is fictional???

Generics and Java
For example
Integer extends Number — True
By Covariance Integer[] extends Number[]
Hence this is legal:
Number[] nArray = new Number[10];
Integer[] iArray = nArray;
can put integers into iArray and it is guaranteed to be fine with nArray
See ArrayCov_java
point when passing into methods covariant type inherit just like their base types. But this can cause issues at run time.
generics are NOT covariant It would break type saftey
For instance consider ArrayList
ArrayList<Integer> ai = new ArrayList<>();
ArrayList<Number> an = ai; // WILL NOT COMPILE
ln.add(Double.doubleValue(2.2));
See also Cov1_java
(note arrays actually have the same issue)
Generics with wildcards
see covar_java
see Wildcard_java
ArrayList<? extends Number>
ArrayList<?>
ArrayList<*>
Wildcards can be handy
limit a function to taking an array list that contains anything that extends number (you need it here because generics are NOT covariant)
But wildcards result in other issues, specifically immutability.
See Immut_java

Type Erasure

the cause of the “R[] arr = (R[]) new Object[100])” problem

generics are known only by compiler, they are “erased” after compile so all of that info is gone at runtime.

see Erasure_java

EG

ArrayList<String> ss = new ArrayList<>();

eventually gets translated to

ArrayList ss = new ArrayList();

So at run time, anything that the compiler let pass is OK. It could cause runtime issues.

Erasure also causes things that might see legal to NOT be legal. For instance

public class JavascriptNumber implements Comparable<String>, Comparable<Number> { …}

does not work because compiler reduces this to

public class JavascriptNumber implements Comparable, Comparable { …}

Generics in Go (topic 8)