Introduction (1 of 2)

- Go was designed by PL veterans: Robert Griesemer, Rob Pike, and Ken Thompson, at Google, in 2009. Griesemer worked with Niklaus Wirth on Pascal. Pike and Thompson worked on Unix projects at Bell Labs.
- Go is an Algol-class PL: imperative, block-structured, statically scoped, strongly typed, and statically typed.
- Local variables can use type inference. Formal parameters must have explicit types.
- Go is an object-oriented PL, but in a odd way (examples follow).
- Go has pointers, but no pointer arithmetic.
- Go *will* have generics, soon.

Introduction (2 of 2)

- Like C and C++, Go compiles to machine code. Go is one of their few competitors. However, Go has garbage collection!
- Memory for variables is allocated on the stack, unless escape/lifetime analysis determines that it must be allocated on the heap.
- Go has builtin container types: arrays/slices and maps.
- Go is a higher-order PL: function literals (i.e., lambdas), closures, and early binding.
- Go is a concurrent PL: `go` and `select` statements, and channels.

Simple Example

- Here's a Go example, compared to C:

  `pub/sum/go/sum.go`

  `pub/sum/c/sum.c`
- Lowercase names denote "private."
- A definition's type/identifier order is swapped (like Pascal).
- A function definition can have a return variable (like Pascal).
- An array/slice is "smarter."
- A `for` is the only kind of loop, but it is very flexible.
- Notice: packages, library I/O, type inference, assignment (`:=` versus `=`), and semicolons.

Concurrency Example

- Here's a silly multithreaded Go example, compared to Java:

  `pub/sum/go/sumgo.go`

  `pub/sum/java/SumThread.java`
- Go threads are called *goroutines*.
- Channels provide statically typed, atomic, and either unbuffered or buffered communication between goroutines.
- Go's builtin function `make` allocates a new channel (or array slice, or map).
- Go also has a builtin function `new`, which is like Java's `new` or C's `malloc`, for other types of memory allocation.

Object Orientation (1 of 2)

- Go is OO, but you wouldn't know, because it doesn't show. The Language Specification's Table of Contents contains neither the word "class" nor "object."
- Nevertheless, these Go keywords can make it happen: `package`, `type`, `interface`, and `struct`. We also need to learn Go's notions of "method set," "embedding," and "receiver."
- Here's a, OO Go example, versus Java:

  pub/etc/Student/OO.java

  pub/etc/student/main.go

Object Orientation (2 of 2)

- A `Student` is-a `Person`. The files of an inheritance hierarchy should be in one package (e.g., `person`). This allows access to unexported (i.e., lowercase names).
- An embedded interface or struct type denotes the union of the method or member sets. Effectively, this allows multiple inheritance. Name collisions must have identical signatures.
- A `Person` has-a `name`. If a structure member, variable, or formal parameter needs to hold an "object reference," its type should be an interface type, not a structure-pointer type.

Another Concurrency Example

- A computer-science student should learn a new sorting algorithm every week. ☺
- Here's a mappy-reducey implementation of one, named Sleep Sort:

  `pub/etc/sleepsort/SleepSort.java`

  `pub/etc/sleepsort/sleepsort.go`
- Must the output array be written to atomically?
- Must the input array be read from atomically?
- How might you change the program, to show that?