Alias Declarations for Go

A proposal

GopherCon 2016

Robert Griesemer
gri@golang.org

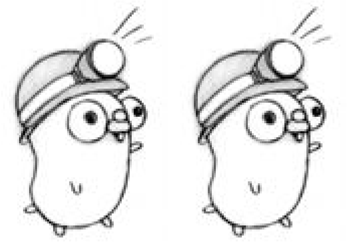
Motivation

1. Suppose we have packages L and C, and C imports L.
2. We decide to refactor L into L and L_1.
3. We must update C to import L and L_1 as needed, *simultaneously*.
4. No big deal!

Bigger motivation

1. Suppose we have packages L and C_1, C_2, ...C_n and n is very large (say, 100, 1000,...).
2. We decide to refactor L into L and L_1.
3. We must update all C_i to import L and L_1 as needed.
4. We may *not* be able to update all C_i *simultaneously*.
5. We may break the continuous build. Big deal!

Real issue in large software environments such as at Google.



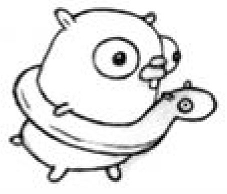
Incremental update to the rescue

- If we could update C_i *incrementally*, problem goes away.
- If we can leave “forwarding” declarations in L for the objects (consts, types, vars, funcs) that moved to L_1, we don’t need to update all C_i at the same time.
- But can we?

Easy for constants and functions

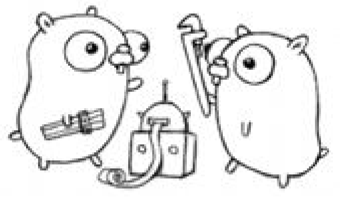
Idea: Leave “forwarding declarations” in old package.

```
package L
import "L1"
const X = L1.X   // L.X and L1.X are the same
var F = L1.F     // L1.F is a function
func G(args…)   { L1.G(args…) }
func H(args…) T { return L1.H(args…) }
```



Not so easy for variables

1. **`package L1; import "L"; var Vptr = &L.V`**
2. Update C_i incrementally to use **`(*L1.Vptr)`**.
3. Move **`L.V`** to **`L1.V`**.
4. Update C_i incrementally again to use **`L1.V`**.
5. Remove declaration of **`L1.Vptr`**.
6. Avoid import cycles!



Not possible for types

Regular type declaration creates a new type:

```
type T L1.T
```

T is not identical to L1.T
T has no methods

There is no work-around.

Want: Alias declarations

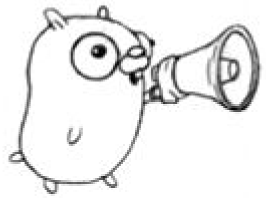
Idea: Extend const notation to all declarations.

```
package L
import "L1"
const X = L1.X
type  T = L1.T  // T is alias for L1.T
var   V = L1.V  // V is initialized to L1.V
func  F = L1.F  // F is alias for L1.F
```

Proposal

Use new token **=>** for alias declarations.

```
package L
import "L1"
const X => L1.X
type  T => L1.T  // T is alias for L1.T
var   V => L1.V  // V is alias for L1.V
func  F => L1.F  // F is alias for L1.F
```



Based on feedback after GopherCon, this slide was adjusted to use of => rather than -> .

Rules

An alias declaration declares an alternative name, the *alias*, for a constant, type, variable, or function, referred to by the rhs of the alias declaration. The rhs must be a (possibly package-qualified) identifier; it may itself be an alias, or it may the original name for the aliased object.

An alias denotes the aliased object, and the effect of using an alias is indistinguishable from the effect of using the original; the only difference is the name.

Examples

```
const π => math.Pi // same as: const π = math.Pi
type T => L1.T
var (
    A => L1.A
    B => L1.B
)
type Op => func(x, y complex128) bool // invalid
type Short => VeryLongNameThatIsHated
```

Based on feedback after GopherCon, we may want to disallow alias declarations for locally declared names.

Syntax changes

```
AliasSpec = identifier "=>" [PackageName "."]
            identifier .
ConstSpec = ...                  | AliasSpec .
TypeSpec  = identifier Type      | AliasSpec .
VarSpec   = ...                  | AliasSpec .
FuncDecl  = "func" FunctionName Signature [Body]
            | "func" AliasSpec .
```

Consequences

Compiler needs to be updated.

(Probably) no changes to linker, package reflect.

go/* libraries and their clients all need to be updated.

New opportunities for misuse...

Unknowns

- Should alias declarations be restricted?

  (aliases for imports only, aliases at top-level only?)
- Are there unintended consequences?
- Is there a better solution?

Summary and next steps

Proposed alias declarations enable incremental refactoring in large-scale software environments.

Significant but backward-compatible language change.

Proposal will go through regular proposal process.

Want community input.

Details: https://github.com/golang/go/issues/16339

Thank you!