

# COMP-520 – GoLite project

Vincent Foley-Bourgon

Sable Lab  
McGill University

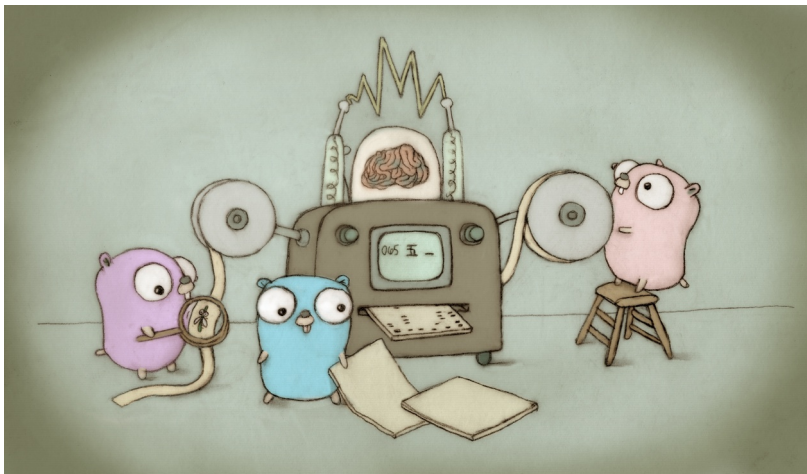
Winter 2020

# Agenda

- ▶ Overview of Go
- ▶ Why Go for a compiler class?
- ▶ GoLite

Feel free to ask questions at any time.





Renee French, licensed under CC 3.0 Attributions

# Go

- ▶ Created by Rob Pike, Ken Thompson and Robert Griesemer
- ▶ Google employees
- ▶ Not a Google project like Gmail; open source
- ▶ Initial release in 2009
- ▶ 1.0 release in 2012

# Motivation

- ▶ Simplify development

# Motivation

- ▶ Simplify development

```
class AbstractSingletonProxyFactoryBean { ... }
```

# Motivation

- ▶ Simplify development

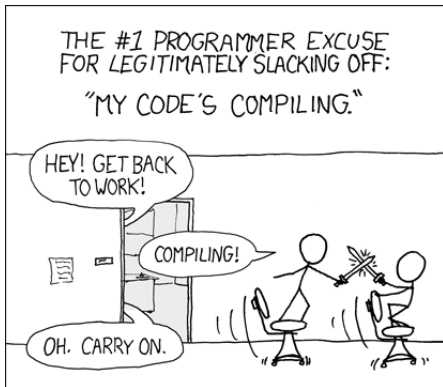
```
class AbstractSingletonProxyFactoryBean { ... }
```

- ▶ Built-in concurrency support



# Motivation

- ▶ Simplify development  
class AbstractSingletonProxyFactoryBean { ... }
- ▶ Built-in concurrency support
- ▶ Faster compilation



# Features

- ▶ Imperative
- ▶ Goroutines and channels
- ▶ Interfaces and methods
- ▶ Closures
- ▶ defer
- ▶ Maps and slices
- ▶ Multiple return values
- ▶ Module system
- ▶ Garbage collection
- ▶ Optional semi-colons (tricky scanner!)

# Notable missing features

- ▶ User-defined parametrized types (source of 95% of all Go arguments online)
- ▶ Exceptions
- ▶ Classes and inheritance
- ▶ Operator overloading

# Example Go program

```
package main

import "fmt"

func fib(n int) int {
    a, b := 0, 1
    for i := 0; i < n; i++ {
        a, b = b, a+b
    }
    return a
}

func main() {
    var f int = fib(42)
    fmt.Println(f)
}
```

# Who uses Go?

- ▶ Google
- ▶ GitHub
- ▶ PayPal
- ▶ Twitch
- ▶ Dropbox
- ▶ Netflix
- ▶ New York Times
- ▶ Many others <sup>1</sup>

Extremely quick adoption!

---

<sup>1</sup><https://github.com/golang/go/wiki/GoUsers>

## Who uses Go?

The authors expected Java and C++ programmers to be the primary Go audience.

In actual fact, Go is more popular with Python, Ruby and other dynamically typed languages programmers.

Why?

# Who uses Go?

The authors expected Java and C++ programmers to be the primary Go audience.

In actual fact, Go is more popular with Python, Ruby and other dynamically typed languages programmers.

Why?

- ▶ Better performance
- ▶ Static typing
- ▶ Good concurrency support
- ▶ Good libraries and tools
- ▶ Can deploy a single binary file

## Useful addresses

- ▶ `http://golang.org`
- ▶ `http://play.golang.org`
- ▶ `http://golang.org/ref/spec`



Why Go for a compiler  
class?

# Why use Go for a compiler class?

Useful and popular

It is more fun to write a compiler for a language that is alive and kicking than for a made-up language (minilang) or for a dead language (Pascal).

Writing a compiler forces you to really learn the language, a nice addition on your C.V.!

# Why use Go for a compiler class?

Simple language

Go is simpler than a lot of other popular languages such as Java or C++.

Go is surprisingly quick to learn.

Not nearly as tricky as MATLAB, JavaScript or PHP.

# Why use Go for a compiler class?

Detailed online specification

You can find pretty much everything you need to know about Go on a single page: <http://golang.org/ref/spec>

The syntax is described in EBNF notation.  
(Warning! Ambiguous!)

Less specification work for the T.A. ;-)

# Why use Go for a compiler class?

Encompasses all the classical compiler phases

The things you learn in class and from reading the textbook apply to writing a Go compiler. It doesn't have specialized phases like pre-processing or macro expansion.

# Why use Go for a compiler class?

Go is open source

Parser used to be written with bison (now hand-written)

The old sources of the parser can be found on Github (e.g. 1.2 release tag)

You can look, **do not copy/paste!**

# Why use Go for a compiler class?

Your work is publishable!

Not only will you learn some Go for this class, but you can publish the complete compiler on GitHub after the class and include it on your C.V.!

# Why use Go for a compiler class?

## External resources

We know that previous year's submissions are available online. We have 3 requirements for this class:

1. You must come up with your own solutions; any inspiration that comes from other sources must be reported and may not replace your own work.
2. No copying from previous years, even with citation; no code/tests may be copied even in part (1 line is too many).
3. No grading material may be used at any point, under any circumstance, nor may it be published.

You are responsible for ensuring all material you use is permitted. If in doubt, please ask!



GoLite



# Features

- ▶ Imperative
- ▶ Goroutines and channels
- ▶ Interfaces and methods
- ▶ Closures
- ▶ defer
- ▶ Maps and slices
- ▶ Multiple return values
- ▶ Module system
- ▶ Garbage collection
- ▶ Optional semi-colons

# Features

- ▶ Imperative
- ▶ Goroutines and channels
- ▶ Interfaces and methods
- ▶ Closures
- ▶ `defer`
- ▶ Maps and slices
- ▶ Multiple return values
- ▶ Module system
- ▶ Garbage collection
- ▶ Optional semi-colons

## Is this still Go?

- ▶ You have a few weeks to build the compiler (took 2 years before first Go release)
- ▶ It still is a lot of work (likely more than you ever put in a class)
- ▶ You can add more features when the course is finished :)

# Lexical syntax

	Go	GoLite
Encoding	UTF-8	ASCII
Number precision	Arbitrary	Fixed
Integers	255, 0377, 0xff	255, 0377, 0xff
Floats	0.12, .12, 12.	0.12, .12, 12.
Imaginary	3i	No thanks
Strings	"Chrono\n"	"Marle\n"
Raw strings	'Lucca\n'	'Ayla\n'
Keywords	Bunch of 'em	Slightly more
Line comments	// Sabin	// Edgar
Block comments	/* Celes */	/* Locke */
Semicolons	Optional	Optional

# Basic types

**int**

**float64**

**bool**

**rune** (char)

**string**

uint8

uint16

uint32

uint64

int8

int16

int32

int64

float32

complex64

complex128

byte

# General structure

```
// Go structure
```

```
// package declaration
```

```
// import statements
```

```
// vars, consts, types, functions
```

# General structure

```
// GoLite structure
```

```
// package declaration
```

```
// vars, types, functions
```



# Declarations

In Go, top-level declarations can be in any order

In GoLite, declarations must come before their first use

```
// Valid in Go; invalid in GoLite
var x int = max(y, 32)
var y = 42

func max(a, b int) int {
    if a > b {
        return a
    } else {
        return b
    }
}
```

# Variable declarations

```
var x1, x2 int           // implicitly initialized to 0
var y int = 12
var z = 24
```

# Variable declarations

```
var x1, x2 int           // implicitly initialized to 0
var y int = 12
var z = 24
```

```
var (
    x1, x2 int
    y int = 12
    z = 24
)
```

GoLite should support all of these.

# Constant declarations

GoLite won't support constant declarations.

# Type declarations

```
type natural int
type real float64

type (
    point struct {
        x, y, z real
    }
)
```

# Function declarations

```
// Allowed in GoLite  
func f(a int, b int) int {  
    ...  
}
```

```
// Allowed in GoLite  
func f(a, b int) int {  
    ...  
}
```

```
// Not allowed in GoLite  
func f(int, int) int {  
    ...  
}
```

- ▶ GoLite functions should always have a body.
- ▶ We'll allow zero or one return value.

# Statements

## Declarations

- ▶ Variables and types can be declared within functions.
- ▶ Short variable declaration allowed within functions.

```
func demo() {  
    type number int  
    var x int = 12  
  
    best_ff := 6  
}
```

# Statements

## Loops

- ▶ All loops use the `for` keyword
- ▶ No parentheses, mandatory braces
- ▶ GoLite should not support *for/range* loops

```
// Infinite loop
for {
    ...
}
```

```
// ‘‘While’’ loop
for x < 10 {
    ...
}
```

```
// ‘‘For’’ loop
for i := 0; i < 10; i++ {
    ...
}
```



# Statements

## Loops

We'll support unlabelled `break` and `continue` in loops

# Statements

## If

- ▶ No parentheses, mandatory braces

```
if x == 0 {  
    ...  
}
```

```
if x < 0 {  
    ...  
} else {  
    ...  
}
```

```
if x < 0 {  
    ...  
} else if x > 0 {  
    ...  
} else {  
    ...  
}
```

# Statements

## Switch

- ▶ Allows expressions in cases
- ▶ No explicit break (although it may be used)

```
switch x {  
  case 0, 1, 2: println("Small")  
  default: println("Other")  
}
```

```
switch { // Same as switch true  
  case x < 0: println("Negative")  
  case x > 0: println("Positive")  
  default: println("Zero")  
}
```

# Expressions

Literals

42, 3.14, "Go", 'H'

Identifiers

x, my\_dog, Alakazou

Unary expressions

!x, +y, -(a\*b), ^0

Binary expressions

a || b, 3 + x, 1 << 12

Function calls

fib(42), max(0, 1)

Casts

int(3.4), mytype(x)

Indexing

slice[0]

Selection

point.x

# Built-ins

In Go:

- ▶ Look like function calls
- ▶ Not reserved keywords
- ▶ Can accept a type as a first parameter (`make([]int, 4)`)
- ▶ Can be polymorphic (`append()`)

Real tricky to parse function calls, casts and builtins nicely

# Built-ins

In GoLite:

- ▶ Reserved keywords to make parsing easier
- ▶ Only a subset (`print`, `println`, `append`, `len`, `cap`)
- ▶ Limited functionality

## References

- ▶ Go presentation:  
`http://www.youtube.com/watch?v=rKnDgT73v8s`
- ▶ Gopher: `http://golang.org/doc/gopher/frontpage.png`
- ▶ Gopher + helmet: `http://golang.org/doc/gopher/pencil/gopherhelmet.jpg`
- ▶ Xkcd, compiling: `http://xkcd.com/303/`

# Advice

- ▶ This is a project that takes a lot of time: start milestones early!
- ▶ Pick an implementation language that you know well enough to not get painted into a corner.
- ▶ Be careful with your AST design, it's extremely important.
- ▶ Don't be afraid of asking questions and using the Facebook group.



# Gophers!

Thanks Google :)

