



Using the Go Programming Language in Practice

Erik Westrup & Fredrik Pettersson

Department of Computer Science, Lund University
Axis Communications, Sweden

May 28, 2014

Supervisors:

Jonas Skeppstedt <jonas.skeppstedt@cs.lth.se> (also Examiner)

Mathias Bruce <mathias.bruce@axis.com>

Robert Rosengren <robert.rosengren@axis.com>

Authors

Erik Westrup

<erik.westrup@gmail.com>

M.Sc. in Computer Science &
Engineering, Lund University
2009–

From: Ljunghusen, Sweden

\$EDITOR=vim

\$BROWSER=firefox+pentadactyl
colorscheme=solarized



Fredrik Pettersson

<fredrik.pettersson.89@gmail.com>

M.Sc. in Computer Science &
Engineering, Lund University
2009–

From: Halmstad, Sweden

\$EDITOR=sublime2

\$BROWSER=chromium
colorscheme=solarized



Table of Contents

- Introduction
- Approach
- Discussion
- Conclusions

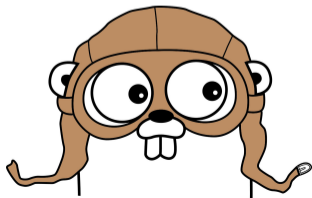
Introduction

Introduction to Go

- Designed by: Ken Thompson, Rob Pike et al. at Google
- Appeared in 2009
- Open source

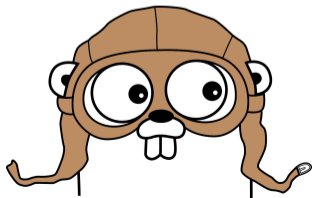
Introduction to Go

- Designed by: Ken Thompson, Rob Pike et al. at Google
- Appeared in 2009
- Open source



Introduction to Go

- Designed by: Ken Thompson, Rob Pike et al. at Google
- Appeared in 2009
- Open source
- Meets large-scale dev. problems including:
 - long build times
 - uncontrolled dependencies
 - hard to understand code



Purpose & Goals

- Review of Go + tools
- Go for embedded programming?
- How to review a language?

Problem Formulation

Problem Formulation

- Go
 - Easy to learn?
 - Mature?
 - Future?

Problem Formulation

- Go
 - Easy to learn?
 - Mature?
 - Future?
- Building & Compiling
 - Easy?
 - Fast?
 - Avoid tools like Makefiles?
 - Cross-compilation
 - C-integration?

Problem Formulation

- Go
 - Easy to learn?
 - Mature?
 - Future?
- Development tools?
- Software Product & Development Qualities?
- Building & Compiling
 - Easy?
 - Fast?
 - Avoid tools like Makefiles?
 - Cross-compilation
 - C-integration?

Approach

Language design goals

Language design goals

Features

- Simple specification
- Easy to understand code
 - C-like syntax
 - Compiled & statically typed
 - Type inference

Language design goals

Features

- Simple specification
- Easy to understand code
 - C-like syntax
 - Compiled & statically typed
 - Type inference
- OOP: interfaces, structs & composition
- Concurrency: goroutines & channels
- Garbage collection

Language design goals

Features

- Simple specification
- Easy to understand code
 - C-like syntax
 - Compiled & statically typed
 - Type inference
- OOP: interfaces, structs & composition
- Concurrency: goroutines & channels
- Garbage collection

Lack of Features

- Type-orientation: inheritance
- Generics
- Exceptions
- Fat standard library
- Pointer arithmetic

Hello, world

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello , World!")
7 }
```

Types

```
1  var i int
2  var s []bool
3  var m map[int] string
4  f := 3.14
```

Declaration order

C, inside-out:

```
typedef char *(*(func_collection[1]))(int *);
```

Declaration order

C, inside-out:

```
typedef char *(*(func_collection [1]))(int *);
```

Go, left-to-right:

```
type funcCollection [1](func(*int) string)
```

Multiple return values & errors

```
1 func check(input string) (result string, err error) {  
2     err = ...  
3     result = ...  
4     return  
5 }
```

Structs & Method receivers

```
1 type Object struct {  
2     name string  
3 }  
4  
5 func (o Object) String() string {  
6     return fmt.Sprintf("Object %s", o.name)  
7 }
```

Interfaces

```
1 type Stringer interface {  
2     String() string  
3 }
```


Composition with struct-embedding

```
1 type B struct {  
2     bData int  
3 }  
4  
5 type A struct {  
6     aData int  
7     B  
8 }  
9  
10 func (b B) operate() {  
11 }
```

goroutines

- Lightweight threads
- Multiplexed into OS threads

```
go work()
```

Channels

- For goroutine communication
- Declared for a given type
- Synchronized or asynchronous
- Replaces semaphores/mutexes

```
1 var channel chan int
2 channel ← 8 // Send
3 i := ←channel // Receive
```

Go tools

```
$ go help
```

Go is a tool for managing Go source code.

Usage: go command [arguments]

The commands are:

build	compile packages & dependencies
fmt	run gofmt on package sources
get	download & install packages & dependencies
test	test packages
...	...

Building

- Easy to use
- Automated
- Smart dependency resolution
- The environment

```
$ go build <package>
```

gc VS gccgo

gc VS gccgo

gc (Go Compiler)

- Default compiler
- Ideas from Plan 9 compilers
- OSes:
Linux, FreeBSD, OS X,
Windows
- Architectures:
i386, amd64, arm

gc VS gccgo

gc (Go Compiler)

- Default compiler
- Ideas from Plan 9 compilers
- OSes:
Linux, FreeBSD, OS X, Windows
- Architectures:
i386, amd64, arm

gccgo

- Front-end for GCC
- OSes:
Linux, BSD, OS X, Windows, IRIX, Solaris
- Architectures:
i386, x86_64, amd64, arm, arm64, mips, alpha, m68k, powerpc, sparc

Cross-compiling with *gccgo*

Building *x-gccgo*

- As easy as building *x-compiler* for C

Cross-compiling with *gccgo*

Building *x-gccgo*

- As easy as building *x-compiler* for C
- Building *x-compiler* is very challenging

Cross-compiling with *gccgo*

Building *x-gccgo*

- As easy as building *x-compiler* for C
- Building *x-compiler* is very challenging
- Helper tools currently problematic
- Built a 4.9 GCC *x-toolchain*.

Cross-compiling with *gccgo*

Building *x-gccgo*

- As easy as building *x-compiler* for C
- Building *x-compiler* is very challenging
- Helper tools currently problematic
- Built a 4.9 GCC *x-toolchain*.

Go + *gccgo*

- Unresolved issue

C-Integration

- Tool: *cgo*
- Go-code:
 - C-code & compiler directives in comments
 - Can use C-symbols
 - Can convert C \longleftrightarrow Go types
- C-code:
 - Can use Go-symbols

C-Integration

- Tool: *cgo*
- Go-code:
 - C-code & compiler directives in comments
 - Can use C-symbols
 - Can convert C \longleftrightarrow Go types
- C-code:
 - Can use Go-symbols
- Problems: Callbacks, pointers, macros etc.

Debugging

- GDB works, bad goroutine & core dump support
- Uncertain GDB future
- Go-specific debugger?

Other tools

Many good tools exists:

- Built-in:
 - Testing
 - Documentation
 - Code formatter
 - Package manager
- Text editors & IDEs
- Convenient tools

Discussion

Design Goals

“Less is exponentially more”
– Rob Pike

How did it turn out?

- 👍 Well afterthought language
- 👍 Flexible OOP
- 👎 👍 Generics: Necessary?
- 👍 Awesome concurrency model

Building





- 👍 Zero project startup cost
- 👎 Less control & understandability
- 👍 Better in the long run
- 👎 Warnings = Errors

Developer Tools

gc VS gccgo





gc VS gccgo

gc





-  Up-to-date
-  Fast
-  Few architectures
-  Only static linking

gc VS gccgo

gc

-  Up-to-date
-  Fast
-  Few architectures
-  Only static linking






gccgo

-  Supports many platforms
-  Slower
-  Dynamic linking
-  Different release schedule

Cross-Compilation

- 👉 Uncertain Go + gccgo combination
 - LLVM is the future?

C-Integration

-  Works well in simple cases
-  Code in comments
-  Locating errors
-  Wrapper-functions
-  Slower builds

Other Tools

- Package manager
 - 👍 Simplifies development
 - 👎 No version support
- 👍 Testing & documentation
- 👎 State of debugging

Community

- 👍 Contributes to Go & tools
- 👍 Friendly & helpful
- 👍 Lead developers participate

Future of Go

- 👍 Stabilization/optimization of tools
- 👉 Embedded programming: mild interest

Future of Go

- 👍 Stabilization/optimization of tools
- 👎 Embedded programming: mild interest
 - Google goes, Go goes?
 - Competing languages?

Worth Doing a Project?

Worth Doing a Project?

- Yes! (if relevant & restricted)
- Reveals the true nature

Conclusions

Summary

- 👍 Easy to learn & adopt
- 👍 Suitable for large-scale projects
- 👍 Concurrency

Summary

- 👍 Easy to learn & adopt
- 👍 Suitable for large-scale projects
- 👍 Concurrency
- 👎 Complicated C-integration
- 👎 Cross-compilation with go + gccgo
- 👍 Reviewing languages → larger project

Summary

- 👍 Easy to learn & adopt
- 👍 Suitable for large-scale projects
- 👍 Concurrency
- 👎 Complicated C-integration
- 👎 Cross-compilation with go + gccgo
- 👍 Reviewing languages → larger project

Go for Embedded?
→ Wait a while

Summary

- 👍 Easy to learn & adopt
- 👍 Suitable for large-scale projects
- 👍 Concurrency
- 👎 Complicated C-integration
- 👎 Cross-compilation with go + gccgo
- 👍 Reviewing languages → larger project

Go for Embedded?
→ Wait a while



Future Research

- Effectiveness of goroutines
- gc & gccgo optimization & compilation speed comparison
- Debugging
- Memory profiling + C-integration

Have a couple of hours to spare? Learn Go!

tour.golang.org

Have a couple of hours to spare? Learn Go!

tour.golang.org

Download our thesis @ bit.ly/go-thesis

