Introduction to Programming in Go

1. Getting started with Go

Preliminaries

Introductions

1. Your area of expertise.

2. Your current job role and responsibilities.

3. Programming experience.

4. Expectations for the course.

Expectations

1. Go is installed on your computer if you are not using a supplied virtual machine.

2. You can program in a C-type language.

Hello World

Hello World

As required by international law... first program is Hello World

```
// Example 01-01 Hello World

package main

import "fmt"

func main() {
   fmt.Println("Hello World!")
}
```

Compiling and Running

- "go run" compiles and executes a Go program.
- Executable is deleted right after execution.

```
[Module01]$ ls
Ex01-01.go
[Module01]$ go run Ex01-01.go
Hello World!
[Module01]$  ls
Ex01-01.go
```

Building

- "go build" compiles packages and dependecies.
- executable exists after command finishes.

```
[Module01]$ ls
Ex01-01.go
[Module01]$ go build Ex01-01.go
[Module01]$ ls
Ex01-01  Ex01-01.go
[Module01]$ ./Ex01-01
Hello World!
[Module01]$
```

The Go Tool

```
Usage:

        go command [arguments]

The commands are:

        build       compile packages and dependencies
        clean       remove object files
        doc         show documentation for package or symbol
        env         print Go environment information
        fix         run go tool fix on packages
        fmt         run gofmt on package sources
        generate    generate Go files by processing source
        get         download and install packages and dependencies
        install     compile and install packages and dependencies
        list        list packages
        run         compile and run Go program
        test        test packages
        tool        run specified go tool
        version     print Go version
        vet         run go tool vet on packages
```

Go File Structure - Main Package

```
// Example 01-01 Hello World

package main

import "fmt"

func main() {
   fmt.Println("Hello World!")
}
```

- packages in Go are libraries or modules.
- each source file begins with a package declaration.
- the main package is not a library but declares the file to be a standalone executable program.

Go File Structure - Imports

```
// Example 01-01 Hello World

package main

import "fmt"

func main() {
   fmt.Println("Hello World!")
}
```

- to use symbols in other packages, the packages must be imported.
- it is a compile time error to import a package that is not used.
- package names serve as name spaces.

Go File Structure - main()

```
// Example 01-01 Hello World

package main

import "fmt"

func main() {
   fmt.Println("Hello World!")
}
```

- the main() function is always in the main package.
- execution of an application begins with a call to main().

Go File Structure - Imported Symbol

```
// Example 01-01 Hello World

package main

import "fmt"

func main() {
   fmt.Println("Hello World!")
}
```

- imported symbols are prefixed with the package name.
- only symbols that start with a capital letter can be imported.
- symbols that start with a lowercase letter are private to a package.

Go File Structure - Multiple Imports

```
// Example 01-02 Hello World

package main

import  (
        "fmt"
        "strings"
)

func main() {
   fmt.Println(strings.ToUpper("Hello World!"))
}
```

Why Go?

Large Scale Software Development

The Go programming language was conceived in late 2007 as an answer to some of the problems we were seeing developing software infrastructure at Google.

The problems introduced by multicore processors, networked systems, massive computation clusters, and the web programming model were being worked around rather than addressed head-on.

Moreover, the scale has changed: today's server programs comprise tens of millions of lines of code, are worked on by hundreds or even thousands of programmers, and are updated literally every day. To make matters worse, build times, even on large compilation clusters, have stretched to many minutes, even hours.

Rob Pike

Rationale for Go

1. Not a "better" programming language or new way of programming.

2. C and C++ produce faster executing code in general

3. Addresses issues of build and delivery for complex and large scale software projects.

4. Utilizes modern technology for efficiency which older languages can not do.

5. Address issues of scale that cannot be addressed with older languages' compile and build technology.

6. Provides a consistent tool set to avoid third party tool mahem.

Complexity Factors

Factors that contribute to complexity in builds

1. Slow compilation and dependency resolution.
2. Lines of code and number of modules.
3. Number of programmers.
4. Multiple third party tools inconsistently used.
5. Different subsets of language used.
6. Older languages unable to use modern hardware efficiently.

Reductionistic Go

Go simplifies for build efficiency and clean code

1. Features that make other languages overly complex are omited.
2. Features producing overly complex compiled code are eliminated.
3. Features that slow compilation are eliminated.
4. Concurrency is a fundamental part of the language.
5. Incorporates remote importing of packages.
6. Supports modern environments and practices (eg. unit testing).

Formatting Go

Go Formatting

Go is formatted much like C or Java except that:

1. Semi-colons are required only to separate statements on the same line – eols ('\n') are used to end a statement.

2. We can not insert eols freely like we can in Java or C.

3. Opening braces cannot be preceded by an eol.

4. Other white space is not significant.

5. "gofmt file" applies proper Go formatting to the file.

6. "gofmt" does not fix errors due to eol issues.

Standard Libraries

Standard Libraries

1. Like other languages, a large number of standard libraries like "fmt" exist.

2. We will be mentioning them and using them from time to time in the code.

3. There is extensive documentation on line at the Go language site.

4. Aside from pointing out specific features of interest, you are expected to explore the libraries on your own initiative.

Lab 1: Getting Started