Program Construction

Objectives

- Define Go program structure
- List the Go identifier rules
- Describe the function of packages and go get

Program Structure

- A Go program is stored in one or more files with the ".go" extension
- Files begin with a package declaration
  - This identifies the package the file is a part of
  - The folder name containing the files must match the package name
- The package declaration is followed by required import declarations
- Imports are followed by a sequence of package-level declarations of types, variables, constants, and functions, in any order

example.go

```go
package main

import (
    "fmt"
    "sailing"
)

func main() {
    var i float32 = 77.5
    var j float32 = 23.3
    var e float32 = 21.7
    var p float32 = 74.3
    fmt.Println("Main area", sailing.CalcM(e, p))
    fmt.Println("Foretriangle", sailing.CalcFT(j, i))
    fmt.Println("Sail Area", sailing.CalcSailArea(e, p, j, i))
}
```

```
Randy@romolack MINGW64 /d/dev/go/example
$ ls -l src/example.go
-rw-r--r-- 1 Randy 197609 322 Apr  8 21:53 src/example.go

Randy@romolack MINGW64 /d/dev/go/example
$ ls -l src/sailing
total 2
-rw-r--r-- 1 Randy 197609  95 Apr  8 18:55 power.go
-rw-r--r-- 1 Randy 197609 400 Apr  8 21:53 sail.go
```

Packages

- Packages in Go act like libraries or modules in other languages
- The source code for a package resides in one or more .go files in a directory whose name ends with the import path
  - prometheus/discovery/discovery.go
  - stored in $GOPATH/src/github.com/prometheur/prometheus/discovery
- The GOPATH environment variable is used to specify the Go Workspace
  - The Go workspace is the directory under which projects and packages are located
  - Even though the GOPATH may be a list of directories, it is generally set to a single folder for all Go code on your machine
  - Subdirectories are searched as needed
- Each package serves as a separate namespace
  - To refer to a function from outside its package, qualify the identifier with the package name
  - discovery.ProvidersFromConfig()
- The comment at the top of a package file serves as the package documentation
  - Only one file in each package should have a package doc comment
  - Extensive doc comments are typically placed in a file of their own called doc.go by convention

prometheus / prometheus

Code | Issues 155 | Pull requests 41 | Projects 0 | Wiki

Branch: master | prometheus / discovery / discovery.go

Gouthamve Fix Map Race by Moving Locking closer to the Write (#2476)

3 contributors

309 lines (275 sloc) | 8.5

```
GOPATH=/home/user/go

/home/user/go/
    src/
        foo/
            bar/                (go code in package bar)
                x.go
            quux/               (go code in package main)
                y.go
    bin/
        quux                    (installed command)
    pkg/
        linux_amd64/
            foo/
                bar.a           (installed package object)
```

```
1   // Copyright 2016
2   // Licensed under
3   // you may not use
4   // You may obtain
5   //
6   // http://www.apac
7   //
8   // Unless required
9   // distributed under the License is distributed on an "AS IS" BASIS,
10  // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
11  // See the License for the specific language governing permissions and
12  // limitations under the License.
13
14  package discovery
15
16  import (
17          "fmt"
18          "sync"
```

Package Names

- When a package is imported the package name becomes an accessor for the contents
  - import “test"
- The package name should be:
  - Short
  - Concise
  - Evocative
- By convention packages are given lower case single-word names
  - No need for underscores or mixedCaps
  - Err on the side of brevity, everyone using your package will be typing that name
- Don't worry about collisions
  - The package name is only the default name for imports
  - The package name need not be unique across all source code
  - In case of a collision the importing package can choose a different name to use locally
- Another convention is that the package name is the base name of its source directory
  - The package in src/encoding/base64 is imported as "encoding/base64" but has name “base64”
- Don't use the ‘import .’ Notation
  - This can simplify tests that must run outside the package they are testing, but should otherwise be avoided
- Consider the package name when naming package elements
  - The buffered reader type in the bufio package is called Reader, not BufReader
  - Users see it as bufio.Reader
  - bufio.Reader does not conflict with io.Reader
  - Use the package structure to help you choose good names
  - once.Do; once.Do(setup) reads well and would not be improved by writing once.DoOrWaitUntilDone(setup)
    - Long names don't automatically make things more readable
    - A helpful doc comment can often be more valuable than an extra long name

Package import and initialization

- It is an error to refer to an external element without importing its package
- It is an error to import a package and not refer to it
- The goimports tool will automatically configure an application's import statements
  - Install via package manager or with go get
  - go get golang.org/x/tools/cmd/goimports
- Packages initialization
  - Package variables are initialized in the order declared
    - dependencies are resolved first however
  - Files in a Package are processed in lexical order
- Any file may contain any number of init functions
  - func init() { /* ... */ }
  - init functions can't be called or referenced
  - init functions are automatically executed when the program starts
    - in the order in which they are declared

```
user@ubuntu:~/go/src/lab03$ cat lab03.go
package main

func main() {
    fmt.Println("Hi")
}

user@ubuntu:~/go/src/lab03$ goimports lab03.go > lab03b.go
user@ubuntu:~/go/src/lab03$ cat lab03b.go
package main

import "fmt"

func main() {
        fmt.Println("Hi")
}
```

Scope

- Syntactic block
  - A sequence of statements enclosed in braces
  - Identifiers declared within a syntactic block are not visible outside that block
- Lexical blocks
  - The entire source code
    - the universe block
  - Each package
  - Each file
  - Each for
  - Each if
  - Each switch
    - Also for each case in a switch or select statement
  - Each explicit syntactic block
- A declaration's lexical block determines its scope
  - Builtins live in the Universe Block
  - Declarations outside any function (at package level) can be referred to from any file in the same package
  - Imported packages are declared at the file level, so they can be referred to from the same file
  - Local declarations can be referred to only from within the same block

Inner declarations shadow (hide) outer declarations

example.go

```go
package main

import "fmt"
import "sailing"

func main() {
    fmt.Println(sailing.CalcFT(12.7, 18.9))
}
```

sail.go

```go
package sailing

//CalcM returns the main sail area
func CalcM(e float32, p float32) float32 {
    return e * p / 2
}

//CalcFT returns the fore triangle (jib sail area)
func CalcFT(j float32, i float32) float32 {
    return j * i / x
}

//CalcSailArea returns the total sail area
func CalcSailArea(e float32, p float32, j float32, i float32) float32 {
    return CalcM(e, p) + CalcFT(j, i)
}
```

power.go

```go
package sailing

const x = 2.0

var y int = 8

func test() float32 {
    return x
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
API server listening at: 127.0.0.1:34218
120.015
```

go get

- The default $GOPATH on *nix is $HOME/go
  - Many store programs there
- The "go get" subcommand allows you to install packages from the internet on your GOPATH
  - $ go get github.com/google/codesearch/index
  - $ go get github.com/petar/GoLLRB/llrb
  - The specified projects are downloaded and installed into $HOME/go
    - src/github.com/google/codesearch/index/
    - src/github.com/petar/GoLLRB/llrb/
    - The compiled packages and their dependencies are placed in pkg/

```
user@ubuntu:~/go/lab01$ ll ../src
total 8
drwxrwxr-x 2 user user 4096 Apr  9 00:23 ./
drwxrwxr-x 4 user user 4096 Apr  9 00:17 ../
user@ubuntu:~/go/lab01$ go get github.com/golang/example/hello
user@ubuntu:~/go/lab01$ ll ../src
total 12
drwxrwxr-x 3 user user 4096 Apr  9 00:23 ./
drwxrwxr-x 6 user user 4096 Apr  9 00:23 ../
drwxrwxr-x 3 user user 4096 Apr  9 00:23 github.com/
user@ubuntu:~/go/lab01$ ll ../src/github.com/
total 12
drwxrwxr-x 3 user user 4096 Apr  9 00:23 ./
drwxrwxr-x 3 user user 4096 Apr  9 00:23 ../
drwxrwxr-x 3 user user 4096 Apr  9 00:23 golang/
user@ubuntu:~/go/lab01$ ll ../src/github.com/golang/
total 12
drwxrwxr-x 3 user user 4096 Apr  9 00:23 ./
drwxrwxr-x 3 user user 4096 Apr  9 00:23 ../
drwxrwxr-x 9 user user 4096 Apr  9 00:23 example/
user@ubuntu:~/go/lab01$ ll ../src/github.com/golang/example/
total 52
drwxrwxr-x  9 user user  4096 Apr  9 00:23 ./
drwxrwxr-x  3 user user  4096 Apr  9 00:23 ../
drwxrwxr-x  3 user user  4096 Apr  9 00:23 appengine-hello/
drwxrwxr-x  8 user user  4096 Apr  9 00:23 .git/
drwxrwxr-x 12 user user  4096 Apr  9 00:23 gotypes/
drwxrwxr-x  2 user user  4096 Apr  9 00:23 hello/
-rw-rw-r--  1 user user 11358 Apr  9 00:23 LICENSE
drwxrwxr-x  2 user user  4096 Apr  9 00:23 outyet/
-rw-rw-r--  1 user user  2634 Apr  9 00:23 README.md
drwxrwxr-x  2 user user  4096 Apr  9 00:23 stringutil/
drwxrwxr-x  2 user user  4096 Apr  9 00:23 template/
user@ubuntu:~/go/lab01$ ll ../src/github.com/golang/example/hello/
total 12
drwxrwxr-x 2 user user 4096 Apr  9 00:23 ./
drwxrwxr-x 9 user user 4096 Apr  9 00:23 ../
-rw-rw-r-- 1 user user  706 Apr  9 00:23 hello.go
user@ubuntu:~/go/lab01$
```

go get tools

- The go get fetching of source code is done by using one of the following tools expected to be found on your system:
  - svn - Subversion, download at: http://subversion.apache.org/packages.html
  - hg - Mercurial, download at https://www.mercurial-scm.org/downloads
  - git - Git, download at http://git-scm.com/downloads
  - bzr - Bazaar, download at http://wiki.bazaar.canonical.com/Download
- For example, git is used for Github, hg is used for Bitbucket, etc.
- For setting proxies for these tools, look here:
  - https://github.com/golang/go/wiki/GoGetProxyConfig

Identifiers

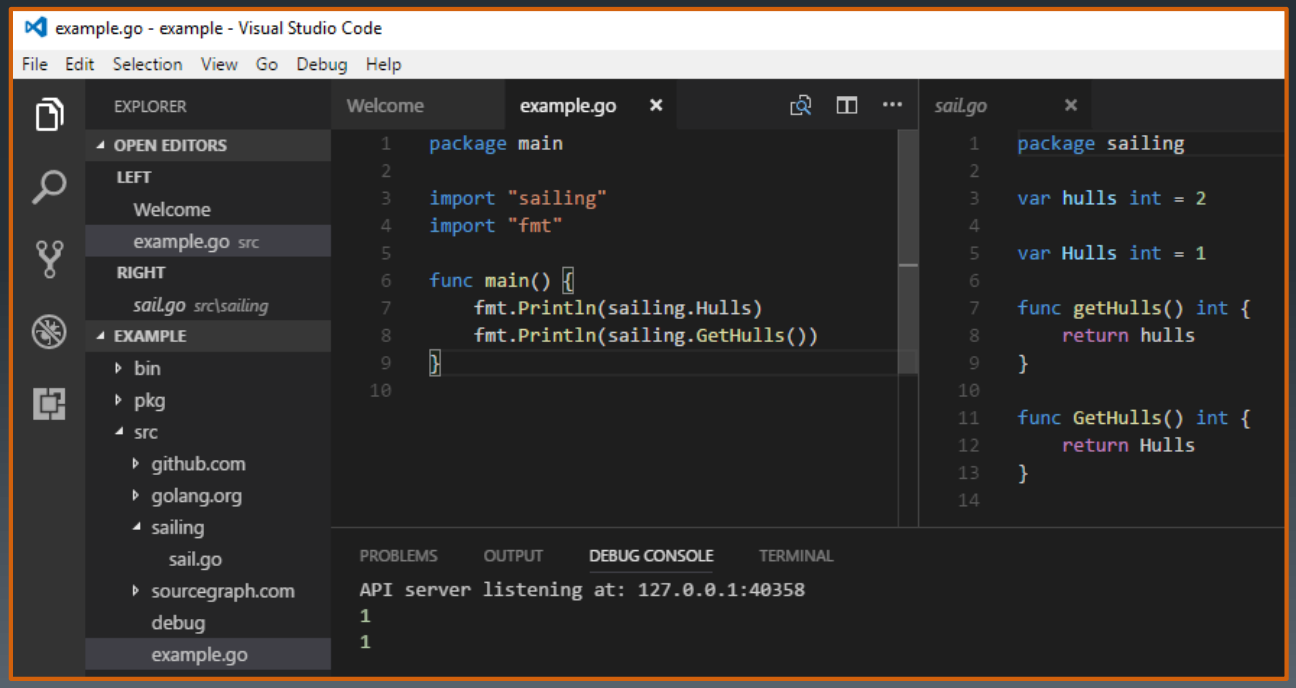
- Go identifiers are used to reference variables, types and other user defined things
- Identifiers are case sensitive
- Identifiers must begin with a Unicode letter or an underbar
  - Unicode divides characters into several major categories: Letter, Mark, Number, Punctuation, Symbol, Separator and Other
- After the first character Names can container any number of letters, underbars and/or digits (there is no limit on name length)
- Idiomatic Go uses camel case
  - Acronyms are always same case (e.g. HTML or html, not Html)
- Keywords cannot be used as identifiers, the 25 go keywords are:

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

- Predeclared names can be used as identifiers in some cases yet should be avoided
  - Constants:
    - true false iota nil
  - Types:
    - int int8 int16 int32 int64 uint uint8 uint16 uint32 uint64 uintptr float32 float64 complex128 complex64 bool byte rune string error
  - Functions:
    - make len cap new append copy close delete complex real imag panic recover

Identifier Visibility

- If an entity is:
  - Declared within a function it is local to that function
  - Declared outside of a function it is visible in all files of the package
    - If the name also begins with an upper-case letter it is visible externally
      - Such identifiers are said to be exported
      - e.g. Println in the fmt package
- Package names are always lower case
- Using sub-packages for name-spacing is generally not recommended
- Idiomatic Go uses short identifiers for small scopes longer and more meaningful identifiers for larger scopes



Summary

- Go program structure
- Packages
- go get
- Go identifier rules
- Go variables
- Go variable scope and lifetime

Lab: Program Construction and Syntax

- Create more complex programs with various identifiers