RX-M Cloud Native Consulting

Go

Data Types

In this lab we will explore Go types and some new Go tools and libraries.

1. Using Go tools

Up to this point we have been getting by with the most basic of Go tools at the command line, the `go` command.

```
user@ubuntu:~$

user@ubuntu:~$ ls -l $(go env GOROOT)/bin
total 28112
-rwxr-xr-x 1 root root 10073055 May 24 11:16 go
-rwxr-xr-x 1 root root 15226597 May 24 11:16 godoc
-rwxr-xr-x 1 root root  3481554 May 24 11:16 gofmt
user@ubuntu:~$
```

Many other useful Go tools are in common use, one is `goimports`.

goimports

The official way to install the standard Go tools is via `go get`. To install `goimports` issue the following command:

```
user@ubuntu:~$ go get -u golang.org/x/tools/cmd/goimports
user@ubuntu:~$
```

The -u flag instructs get to use the network to update the named packages and their dependencies. By default, get uses the network to check out missing packages but does not use it to look for updates to existing packages.

Confirm the goimports command is available:

```
user@ubuntu:~$ which goimports
user@ubuntu:~$
```

Well thats no good, what is going on?

As it turns out, `x/tools` are placed in `$GOPATH/bin` via `go get`, similar to `go install`.

```
user@ubuntu:~$ ls $(go env GOPATH)/bin

goimports  lab-program-construction

user@ubuntu:~$
```

Lets add our local GO binaries to our PATH.

```
user@ubuntu:~$ echo "export PATH=/home/user/go/bin:$PATH" >> .bashrc
```

```
user@ubuntu:~$ source .bashrc
```

Try again:

```
user@ubuntu:~$ which goimports
```

```
/home/user/go/bin/goimports

user@ubuntu:~$
```

Create a working directory for lab-data-types so that we can test out `goimports`.

```
user@ubuntu:~$ mkdir ~user/go/src/lab-data-types

user@ubuntu:~$ cd ~user/go/src/lab-data-types/

user@ubuntu:~/go/src/lab-data-types$
```

Now create a sample program with a missing import:

```
user@ubuntu:~/go/src/lab-data-types$ vim lab-data-types-1.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-1.go
```

```
package main

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

```
user@ubuntu:~/go/src/lab-data-types$
```

Now let use `goimports` to resolve all of the import issues:

```
user@ubuntu:~/go/src/lab-data-types$ goimports lab-data-types-1.go
```

```
package main

import "fmt"

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

```
user@ubuntu:~/go/src/lab-data-types$
```

We can compare the effects of `goimports` against the original file.

```
user@ubuntu:~/go/src/lab-data-types$ goimports lab-data-types-1.go | diff lab-data-types-1.go -

2a3,4
> import "fmt"
>
4c6
<     fmt.Println("Hi")
---
>       fmt.Println("Hi")

user@ubuntu:~/go/src/lab-data-types$
```

- Try diff with the side-by-side comparison `goimports lab-data-types-1.go | diff -y lab-data-types-1.go -`

Let's store the `goimports` generated version.

```
user@ubuntu:~/go/src/lab-data-types$ goimports lab-data-types-1.go > lab-data-types-1b.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-1b.go
```

```
package main

import "fmt"

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

```
user@ubuntu:~/go/src/lab-data-types$
```

By default `goimports` does not modify the original file, it simply outputs the corrected code.

Try running both programs:

```
user@ubuntu:~/go/src/lab-data-types$ go run lab-data-types-1.go

# command-line-arguments
./lab-data-types-1.go:4: undefined: fmt in fmt.Println
user@ubuntu:~/go/src/lab-data-types$
```

```
user@ubuntu:~/go/src/lab-data-types$ go run lab-data-types-1b.go

Hi
user@ubuntu:~/go/src/lab-data-types$
```

The Eclipse GoClipse plugin, the VisualStudio Code Go plugin, and others run `goimports` (often on Save) to clean up code in their IDEs.

Use the `-w` switch to write changes to the original file (aka in-place):

```
user@ubuntu:~/go/src/lab-data-types$ goimports -w lab-data-types-1.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-1.go
```

```
package main

import "fmt"

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

```
user@ubuntu:~/go/src/lab-data-types$
```

Retry the run:

```
user@ubuntu:~/go/src/lab-data-types$ go run lab-data-types-1.go

Hi
user@ubuntu:~/go/src/lab-data-types$
```

Remove the temporary `lab-data-types-1b.go` file.

```
user@ubuntu:~/go/src/lab-data-types$ rm lab-data-types-1b.go

user@ubuntu:~/go/src/lab-data-types$
```

gofmt

Another useful tool is `gofmt`, which is installed with the Go distribution (toolchain) itself.

Give your lab-data-types source some nasty formatting:

```
user@ubuntu:~/go/src/lab-data-types$ vim lab-data-types-2.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-2.go
```

```
package main

import
"fmt"

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

```
user@ubuntu:~/go/src/lab-data-types$
```

Now clean it up with `gofmt`:

```
user@ubuntu:~/go/src/lab-data-types$ gofmt lab-data-types-2.go
```

```
package main

import "fmt"

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

```
user@ubuntu:~/go/src/lab-data-types$
```

Unlike other languages, Go has clearly defined formatting rules. `gofmt` formats Go programs such that they follow these rules. `gofmt` uses tabs (width = 8) for indentation and blanks for alignment. Without an explicit path, `gofmt` processes standard input. Given a file, it operates on that file; given a directory, it operates on all `.go` files in that directory, recursively (files starting with a period are ignored). By default, `gofmt` prints the reformatted sources to standard output.

You can also use `gofmt` in diff mode, try it:

```
user@ubuntu:~/go/src/lab-data-types$ gofmt -d lab-data-types-2.go

diff lab-data-types-2.go gofmt/lab-data-types-2.go
--- /tmp/gofmt616749175 2017-05-22 10:21:03.083814088 -0700
+++ /tmp/gofmt135430506 2017-05-22 10:21:03.083814088 -0700
@@ -1,7 +1,7 @@
 package main

-import
-"fmt"
+import "fmt"

-func main() { fmt.Println(        "Hi")
+func main() {
+       fmt.Println("Hi")
 }
user@ubuntu:~/go/src/lab-data-types$
```

This is useful if you need to create a source code control patch or just want to see the formatting problems in a long file.

We can use the `-w` switch to write changes to the file(s):

```
user@ubuntu:~/go/src/lab-data-types$ gofmt -w lab-data-types-2.go

user@ubuntu:~/go/src/lab-data-types$
```

```
user@ubuntu:~/go/src/lab-data-types$ gofmt -d lab-data-types-2.go

user@ubuntu:~/go/src/lab-data-types$
```

You can build time saving shell aliases with the various go tools. For example:

```
user@ubuntu:~/go/src/lab-data-types$ alias gr='gofmt -w * ; goimports -w * ; go run '
user@ubuntu:~/go/src/lab-data-types$
```

Redo the ugly code.

```
user@ubuntu:~/go/src/lab-data-types$ vim lab-data-types-2.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-2.go
```

```
package main

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

```
user@ubuntu:~/go/src/lab-data-types$
```

Now clean it up with the new alias:

```
user@ubuntu:~/go/src/lab-data-types$ gr lab-data-types-2.go

Hi

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-2.go
```

```
package main

import "fmt"

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

```
user@ubuntu:~/go/src/lab-data-types$
```

To review your alias, just run alias again without setting it.

```
user@ubuntu:~/go/src/lab-data-types$ alias gr

alias gr='gofmt -w *;goimports -w *; go run '

user@ubuntu:~/go/src/lab-data-types$
```

- See other aliases by typing `alias` enter.

2. Working with bits

Using the course presentation as a guide, create a program that performs the following:

- Initializes two uint32 variables to 58 and FFFF7777 respectively
- Display both values in decimal, hex, and binary
- Display the bitwise AND, OR, and XOR of the two values
- Use a for loop to display the setting of the bits sequentially in both uint32 values

3. Reading command line arguments and computing SHA hashes

Enter the following program and run it with three command line parameters:

```
user@ubuntu:~/go/src/lab-data-types$ vim lab-data-types-3.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-3.go
```

```
package main

import "os"
import "fmt"

func main() {
        argsWithProg := os.Args
        argsWithoutProg := os.Args[1:]
        arg := os.Args[3]
        fmt.Println(argsWithProg)
        fmt.Println(argsWithoutProg)
        fmt.Println(arg)
}
```

```
user@ubuntu:~/go/src/lab-data-types$
```

Use `go doc` to explore any features you don't understand.

Enter the following program and run it:

```
user@ubuntu:~/go/src/lab-data-types$ vim lab-data-types-3b.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-3b.go
```

```
package main

import (
        "flag"
        "fmt"
)

func main() {
        wordPtr := flag.String("word", "Hello", "the hello string")
        numPtr := flag.Int("num", 42, "an int with meaning")
        flag.Parse()
        fmt.Println("word:", *wordPtr)
        fmt.Println("num:", *numPtr)
}
```

```
user@ubuntu:~/go/src/lab-data-types$
```

Run the above program with the following arguments:

- --help
- -num 5
- -word bye
- -word bye -num 88
- --word fish
- --num 3 --word fish

Use `go doc` to explore any features you don't understand.

Enter the following program and run it:

```
user@ubuntu:~/go/src/lab-data-types$ vi lab-data-types-3c.go

user@ubuntu:~/go/src/lab-data-types$ cat lab-data-types-3c.go
```

```
package main

import (
        "crypto/sha256"
        "fmt"
)

func main() {
        c1 := sha256.Sum256([]byte("x"))
        c2 := sha256.Sum256([]byte("X"))
        fmt.Printf("%x\n%x\n%t\n%T\n", c1, c2, c1 == c2, c1)
}
```

```
user@ubuntu:~/go/src/lab-data-types$
```

Use go doc to explore any features you don't understand.

Given the above programs, write a new program that prints the SHA256 hash of its standard input by default but supports a command-line flag to print the SHA384 or SHA512 hash instead.

4. Challenge: Arrays and slices

Create a program that declares an array of strings containing at least 5 motorcycle brands (e.g. Honda, Yamaha, Ducati, BMW, Triumph, Polaris). Display the array to stdout. Test the program.

Next create a function that reverses the array and use it to display the reverse of the motorcycle array.

Congratulations you have completed the lab!!