RX-M / go Private

Branch: master go / modules / functions / rx-m-go-lab-functions.md Find file Copy path

ronaldpetty updated functions lab, normalized file names largely 3707416 on Sep 8

3 contributors

134 lines (89 sloc) 4.03 KB

RX-M Cloud Native Consulting

Go

Functions

Go supports first class functions, higher-order functions, user-defined function types, function literals, closures, and multiple return values. This rich feature set enables a functional programming style in a strongly typed language.

1. bufio

Package bufio implements buffered I/O. It wraps an io.Reader or io.Writer object, creating another object (Reader or Writer) that also implements the interface but provides buffering and some help for textual I/O.

The bufio Scanner type provides a convenient interface for reading data such as a file of newline-delimited lines of text. Successive calls to the Scan method will step through the 'tokens' of a file, skipping the bytes between the tokens. The specification of a token is defined by a split function of type SplitFunc; the default split function breaks the input into lines with line termination stripped. Split functions are defined in bufio for scanning a file into lines, bytes, UTF-8-encoded runes, and space-delimited words. You can also provide a custom split function.

Scanning stops unrecoverably at EOF, the first I/O error, or a token too large to fit in the buffer. When a scan stops, the reader may have advanced arbitrarily far past the last token. Programs that need more control over error handling or large tokens, or must run sequential scans on a reader, should use bufio.Reader instead.

Enter the following program using the bufio.Scanner:

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

user@ubuntu:~$



user@ubuntu:~$ mkdir -p ~user/go/src/lab-functions

user@ubuntu:~$



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

user@ubuntu:~/go/src/lab-functions$



user@ubuntu:~/go/src/lab-functions$ vim bikes.go
user@ubuntu:~/go/src/lab-functions$ cat bikes.go

package main

import (
        "bufio"
        "fmt"
        "os"
)

func main() {
        scanner := bufio.NewScanner(os.Stdin)
        var moto string
        for moto != "q" {
                fmt.Print("Input Motorcycle (q to quit): ")
                scanner.Scan()
                moto = scanner.Text()
                if moto != "q" {
                        fmt.Println("Motorcycle entered: ", moto)
                }
        }
}
user@ubuntu:~/go/src/lab-functions$
```

Run and test the program. For example:

```
user@ubuntu:~/go/src/lab-functions$ go run bikes.go

Input Motorcycle (q to quit): Ducati
Motorcycle entered:  Ducati
Input Motorcycle (q to quit): BMW
Motorcycle entered:  BMW
Input Motorcycle (q to quit): Honda
Motorcycle entered:  Honda
Input Motorcycle (q to quit): Yamaha
Motorcycle entered:  Yamaha
Input Motorcycle (q to quit): Polaris
Motorcycle entered:  Polaris
Input Motorcycle (q to quit): q
user@ubuntu:~/go/src/lab-functions$
```

2. Variadics

Using Lab step 1 as a base, create a program that reads in an arbitrary number of motorcycles and then passes them all at once to a single variadic function called BikePrinter, taking `...string`. Have the BikePrinter function display all of the input strings to stdout.

3. First class functions

Create a simple function with the signature `func Italian(string) bool` that takes a bike make string and returns true if it is Italian made (Ducati, Aprilia, Moto Guzzi, ...).

Now modify your Variadic BikePrinter to accept a filter function as an argument. The BikePrinter should only print makes that pass the filter test (where the filter returns true). Run your program passing the Italian() filter function to the BikePrinter as the filter argument.

Create a second filter function, British() that returns true only for British bikes (BSA, Triumph, Norton, ...). Update you program to call the BikePrinter function twice, once with the Italian() filter and once with the British() filter.

Debug and test your program as you go.

4. Challenge: Anonymous functions

Using the program from step 3 above, add a third call to BikePrinter, this time passing an anonymous function that filters for German bikes (BMW, MZ, Sommer, ...). Run and test your code.

Congratulations you have completed the lab!!