calc, a calculator written in Go

Ralph Möritz

September 11, 2012

Contents

1 Summary

This directory contains `calc`, my solution to the AWS challenge in the Go programming language [1]. The challenge was stated as follows:

> You have a text file where each line consists of an operator followed by a colon followed by a comma-separated list of numbers. For each line perform the operation on the numbers and print the results. The valid operators are 'SUM', 'MIN', 'MAX' and 'AVERAGE'.
>
> E.g. for the file

[1] See An Introduction to Programming in Go for a very readable introduction to the basics of the Go language.

SUM: 1, 2, 3 MIN: 4, 3, 2

your program should print:

SUM: 6 MIN: 2

2 Design

`calc` works as follows:

- a REPL reads input one character at a time into a buffer. When a delimiter (colon, comma or newline) is encountered, the buffer is flushed to a string which is subsequently trimmed of leading & trailing whitespace and converted to uppercase.
- The uppercase token is then fed into a state machine which drives the process of building up an expression object from the input.
- Once we have a complete expression consisting of an operator and one or more operands, the REPL evaluates it and prints the result.

3 Prerequisites

To build the code in this directory, you need to have the Go tools installed. You can download binaries for your platform from the Go project's download page. For Windows, simply download and run the MSI installer. For Linux, download the tarball and extract its contents to *usr/local*.

Verify that Go is intalled correctly by launching a shell (command prompt on Windows) and typing `go`. If all is well you should see the following:

```
Go is a tool for managing Go source code.

Usage:

        go command [arguments]

The commands are:

    build       compile packages and dependencies
    clean       remove object files
    doc         run godoc on package sources
```

```
    env         print Go environment information
    fix         run go tool fix on packages
    fmt         run gofmt on package sources
    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

Use "go help [command]" for more information about a command.
Additional help topics:
...
```

4 Testing

You can run a set of unit tests by launching a shell, changing to the `calc` subdirectory and typing `go test`. You should see output similar to the following.

```
go test
PASS
ok      _/x_/personal/code/polyglot/aws-challenge/src/golang/calc       0.091s
```

5 Building

To build the source code in this directory, launch a shell and type `build.bat` (Windows) or `./build` (Linux). If compilation succeeds an executable file named `calc.exe` (Windows) or simply `calc` (Linux) will be created in this directory.

6 Running

You can run `calc` interactively by typing into its input buffer. Your input will be evaluated after you type RET and the result printed to the console as shown in the sample session below.

```
Enter expressions to evaluate followed by a newline. Type "QUIT" to exit.
sum: 124.95, 24.50, 8.99
SUM: 158.440000
min: 92, 11.33, 63.49, 2.9
MIN: 2.900000
quit
Goodbye!
```

You can also use `calc` non-interactively by piping the text to evaluate into stdin. By connecting `calc` to other utilities you can use it in a similar fashion to the venerable Unix calc utility. Below is an example that performs some calculations, sorts the results in descending numeric order and discards all but the top 2 result. This example requires a Unix-like environment to run. On Windows you could download Gow for a basic set of Unix utilities and Bash.

```
$ cat sample_input.txt | ./calc.exe 2>/dev/null | cut -d ' ' -f 2 |
sort -n -r | head -n 2
60.000000
17.500000
```