Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Winter 2019

Lab 4 - Introduction to the Go Programming Language

Demo/Grading

After you finish the exercises, a TA will review your solutions, ask you to run the test harness provided on cuLearn, and assign a grade. For those who don't finish early, a TA will grade the work you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

Prerequisite Reading

Reading Assignment 3, posted on cuLearn (Modules 1 through 3 of *A Tour of Go*)

General Requirements

For those students who already know Go: do not use arrays, structs, pointers, slices or maps. They aren't necessary for this lab.

Your functions must not be recursive. Repeated actions must be implemented using Go's `for` loop.

None of the functions you write should perform console input or produce console output; for example, contain `Printf` or `Println` statements.

Finish each exercise (i.e., write the function and verify that it passes all of its tests) before you move on to the next one. Don't leave testing until after you've written all the functions.

Getting Started

Step 1: Launch a Web browser and go to the Go Play Space website: https://goplay.space/. You can hide the help sidebar by clicking the Settings button, unchecking the Show help sidebar box, then clicking the Settings button again. Delete the sample code in the editor window.

If this website isn't available, you can instead use the Go Sandbox site: https://go-sandbox.com/.

Step 2: Download files power.go, isLeapYear.go, isPrime.go and wire.go from cuLearn.

Exercise 1

Step 1: Launch an IDE (for example, Pelles C) or a text editor (for example, WordPad or any other that editor that can read and write plain-text files.) Use the File > Open command to open power.go. When the Open dialog box appears, you may need to configure it so that files with extension .go are listed. If you're using Pelles C, select All Files (*.*) from the drop-down menu. If you're using WordPad, select All Documents (*.*) from the drop-down menu.

Step 2: Copy/paste the program in power.go into the Go Play Space (or Go Sandbox) editor.

This program has 3 functions:

- `power` calculates and returns an integer base raised to an integer exponent;
- `testPower` is a simple test harness for the `power` function. At this time, you do not need to understand this function's code; the output it produces should be self-explanatory;
- `main` simply calls `testPower`.

Click Run to compile and execute the program. The test harness' output shows that all the tests fail. Reviewing the first few lines, we see that `power(2, 0)` calculates 2^1 instead of 2^0, `power(2, 1)` calculates 2^2 instead of 2^1, and `power(2, 2)` calculates 2^3 instead of 2^2. After reviewing the rest of the output, we conclude that `power(base, n)` actually calculates `base`$^{n+1}$.

Tracing through the body of `power` reveals the case of this flaw: the statement

```
pow *= base
```

is executed one too many times; in other words, `pow` is multiplied by `base` `n+1` times instead of `n` times.

Step 3: Correct the problem by changing the `for` loop condition, `n >= 0`, to `n > 0`. Rerun the program. The test harness' output should now indicate that all the tests pass.

Exercise 2

Open isLeapYear.go in Pelles C/WordPad/your favourite text editor and copy/paste the entire program into the Go editor. (Overwrite all the code from Exercise 1.)

A year with 366 days is called a *leap year*. Usually, years that are divisible by 4 are leap years (for example, 2016 or 2020). However, years that are divisible by 100 (for example, 1900) are not leap years, but years that are divisible by 400 (for example, 2000) are leap years.

Function `isLeapYear` is an incomplete implementation of a function that determines if a year is a leap year:

```
func isLeapYear(year uint) bool
```

The function is passed a single argument, a year (an unsigned integer), and returns a boolean value (`true` or `false`), depending on whether or not the year is a leap year.

Complete the implementation of `isLeapYear` and run the test harness. Use the console output to help you identify and correct any flaws. After verifying that `isLeapYear` passes all the tests, copy/paste it into the editor in which you opened isLeapYear.go, then save the modified file.

Exercise 3

Open a new browser window. (Don't close the window containing your solution to Exercise 2; you'll need to demo it to a TA after you've finished all the exercises.) In the new window, go to the Go Play Space (or Go Sandbox) website.

Open isPrime.go and copy/paste the entire program into the Go editor.

A positive integer *n* is a *prime number* if it is has exactly two positive divisors: *n* and 1. In other words, an integer is defined to be prime if it is not evenly divisible any number other than 1 and itself.

By definition, 1 is not a prime number. 2 is a prime number and is the only even number that is prime. The first few prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19.

Function `isPrime` is an incomplete implementation of a function that determines if a number is a prime number:

```
func isPrime(n int) bool
```

The function is passed a single integer argument and returns a boolean value (`true` or `false`) depending on whether or not the integer is a prime number.

Complete the implementation of `isPrime`. Your function must have `if` statements to handle these special cases:

- `n` is 1, which is not prime;
- `n` is 2, which is prime;
- `n` is any even value other than 2. In this case, `n` is not prime

The function must also contain code that determines if odd values of `n` (other than 1) are prime.

Run the test harness. Use the console output to help you identify and correct any flaws. After verifying that `isPrime` passes all the tests, copy/paste it into the editor in which you opened isPrime.go, then save the modified file.

Exercise 4

Open a new browser window. (Don't close the window containing your solution to Exercise 3; you'll need to demo it to a TA after you've finished all the exercises.) In the new window, go to the Go Play Space (or Go Sandbox) website.

Open wire.go and copy/paste the entire program into the editor.

The resistance *R* of a piece of wire (in ohms, Ω) is given by the formula:

$$R = \rho L/A = 4\rho L/\pi d^2\ \Omega$$

where ρ is the resistivity of the wire and and *L*, *A* and *d* are the length (m), cross-sectional area (m^2) and diameter (m) of the wire. The resistivity of copper wire is $1.7 \times 10^{-8}\ \Omega$ m.

The wire diameter, *d*, is commonly specified by the American Wire Gauge (AWG), which is an integer, *n*, between 1 and 40. The diameter of an AWG *n* wire is given by the formula:

$$d = 0.127 \times 92^{(36-n)/39} \text{ mm}$$

Function `diameter` is an incomplete implementation of a function that is passed a wire gauge and returns the corresponding wire diameter, in mm:

```
func diameter(wireGauge int) float64
```

Function `copperWireResistance` is an incomplete implementation of a function that is passed the length (in m) and gauge of a piece of wire and returns the resistance of that wire:

```
func copperWireResistance(length float64, wireGauge int) float64
```

Remember to account for the units: the value returned by `diameter` is measured in mm, but the length and diameter in the formula for *R* are measured in m.

Complete the implementation of `diameter` and `copperWireResistance`. Hint: Go's `math` package (remember, you have to `import` it) defines the floating point constant `Pi` and function `Pow`:

```
func Pow(x, y float64) float64
```

which returns x^y.

Run the test harness. Use the console output to help you identify and correct any flaws. After verifying that `diameter` and `copperWireResistance` pass all the tests, copy/paste them into the editor in which you opened `wire.go`, then save the modified file.

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the grading/sign-out sheet.
2. Remember to backup your files before you leave the lab; for example, copy them to a flash drive and/or a cloud-based file storage service. All files you've created on the hard disk will be deleted when you log out.

Credits

Exercises 2 - 4 are based on exercises from *Python for Everyone*, 2nd ed., by Cay Horstmann and Rance Necaise, © 2016 John Wiley & Sons.

Last edited: Feb. 5, 2019

- Exercise 4: the Pi symbol in the formula for R didn't appear in the PDF file. Edited the formula to use a Pi symbol that is rendered correctly when the PDF is generated. In the phrase, "where rho is the resistivity", the rho symbol looked like an italicized "Q" when the PDF was displayed. Replaced the Unicode character for rho with a formula (produced with the equation editor) containing only rho.