CMSC 245 – Principles of Programming Languages
Lab#1: Go, code go!

In this lab you will install the Go programming environment as well as the programming language assigned to you on your computer and begin your journey into learning these languages. First, you are directed to the Go website to download and install the Go programming environment on your computer. Also, you can install the Visual Studio Code (VS Code) source code editor (from code.visualstudio.com). You do not have to do this if you are comfortable using the computer science Linux machines, or accessing them remotely. To fire up VS code on a Linux computer in the CS Labs, just use the command "code". You will identify and install an implementation of the programming language assigned to you and use it in PART 4.

You can also write Go programs on the web in Go Playground (https://play.golang.org/). The playground is sufficient for writing basic Go programs. As you learn more features of the language you will to get out of the Go Playground and use a proper Go installation. We will do that next.

PART 1: Installing Go and VS Code

Go to the website: https://golang.org/dl/ and download and run the installer for your computer. You may need to restart your computer.

Test your install using the command:

```
> go version
go version go1.X windows/amd64
```

You will get a specific version and configuration instead of the string above.

PART 2: A First Go Program – Hello, World!

Create a new directory for storing all your Go programs. Navigate to it.

In VS Code, start a new file and enter the following program, as shown below (paying attention to upper/lowercase):

```
package main

import "fmt"

// My first Go program

func main() {
      fmt.Println("Hello, world!")
} // main()
```

Save the program in a file called `hello.go` in the directory you created above.

Next, issue the command:

```
> go run hello.go
Hello, World!
>
```

That's it. Your first Go program!

PART 3: A Second Program.

Enter and run the following program:

```
1  package main
2
3  import (
4     "fmt"
5     "math"
6  )
7
8  func main() {
9     var n = 1.0
10    for n <= 10 {
11          fmt.Printf("%v \t %0.3v\n", n, math.Sqrt(n))
12          n++
13    }
14 } // main()
```

Observe the program carefully. Line numbers are not part of the program. They have been added to locate lines in the discussion below. Here are some salient features:

- **Line 1:** All main programs reside in a package `main`.
- **Lines 3-6:** `fmt` and `math` are packages (we get `Printf()` and `Sqrt()` from them). All package imports have to appear on a separate line as shown.
- **Line 7:** Defines the `main()` function.
- **Line 8:** The variable declaration for n is an implicit declaration of a floating point variable. By default, the type assumed by Go is `float64`. Another way to declare this variable is to explicitly specify the type (try it):

  ```
  var n float64 = 1.0
  ```

  By default, all floating point variables use double-precision (i.e. `float64`). Go also has a single-precision float type: `float32`.
- **Lines 10 -13:** Define a loop. There is no "while" keyword for loops in Go. Go loops all have to be written using `for` loops. There are many ways of writing `for` loops as we will see later. Parentheses are optional in specifying conditions.
- **Line 11:** Uses the `Printf()` function from the `fmt` package. `Printf()` is very similar to the C `printf()`. The `%v` format specification is to print the value as it naturally occurs. You can use `%f` as well. Try it and note the difference.

- **Line 11:** Uses the `Sqrt()` function the `math` package to compute the square root of n.
- What else? Notice, no semi-colons!

Save the program in a file sqRoot.go. Compile and run it as above.

Short variable declarations and counting for loops

Go also allows short variable declarations. That is, you do not need to use the `var` keyword, nor do you have to mention the type, as long as the type is evident from the first use. This is done using the `:=` operator. For example,

```
x := 1.414
```

Above, the type of x is inferred from the expression assigned to it (the value `1.414` which will be treated as `float64`, the default float type).

Like C, Java, etc. Go's for loop can be written as a counting loop:

```
for <initialization>; <condition>; <update> {
     <statements to be repeated>
}
```

Short variable declarations come in handy when writing counting loops. For example, the loop in the `sqRoot.go` program above can be re-written as:

```
for n := 1.0; n <= 10.0; n++ {
     fmt.Printf("%v \t %0.3v\n", n, math.Sqrt(n))
}
```

Rewrite and run the `sqRoot.go` program above to use this loop. Here is another way to write the loop above (just for the sake of completeness and to understand Go's declarations):

```
var n float64
for n = 1.0; n <= 10.0; n++ {
     fmt.Printf("%v \t %0.3v\n", n, math.Sqrt(n))
}
```

The above changes the scope of n. As opposed to the version with implicit declaration, n will not be visible in the program beyond the loop. Is this a good idea?

PART 4: Your Programming Language.

Install an implementation of the programming language assigned to you just as you did for Go. You can use VS Code to write and edit your programs. Repeat PART 2 and PART 3 in your programming language. Observe the differences.