Language Assignment #4: Go

Issued: Thursday, April 1
Due: Tuesday, April 20

Purpose

This assignment asks you to begin using a garbage-collected, concurrent, object-oriented, and imperative programming language named Go. Go has been considered a competitor of the venerable systems-programming duo of C and C++, but it has garbage collection. Go was designed by Robert Griesemer, Rob Pike, and Ken Thompson, at Google, in 2009.

Translator

In our lab, `onyx` is the home-directory file server for its nodes (e.g., `onyxnode01`). There is also a shared directory for "apps" mounted at `/usr/local/apps`. Nodes share a translator for Go, named `gccgo`, which is installed below `/usr/local/apps`, which is a non-standard location.

Due to shared-library constraints, `onyx` cannot execute `gccgo`. It can only be executed by a node.

Due to network constraints, `onyx` can be reached from the public Internet, but a node can only be reached from `onyx`. So, SSH and login to `onyx`, then SSH and login to a node.

An easy way to use `gccgo`, from a node, is to permanently add a line to the end of your `.bashrc` file. To do so, login to a random node, from `onyx`, by executing the script:

```
pub/bin/sshnode
```

Then, execute the script:

```
pub/bin/bashrc
```

Don't change your `$PATH`; just execute the script. Then, logout from the node and login to a node.

The abovementioned `gccgo` is a Go front-end for GCC, the GNU Compiler Collection. As such, it works with GDB, the GNU Debugger, and many other tools. In a different environment, you might want to use the Go toolchain from Google: `https://golang.org/`.

Documentation

Go lecture slides are at:

```
pub/slides/slides-go.pdf
```

Go is demonstrated by:

```
pub/sum/go
pub/etc/sleepsort
```

Links to manuals are at:

```
http://cs.boisestate.edu/~buff/pl.html
```

Go is too new to be described in our textbook.

Assignment

Begin, by porting the simple banking application at:

```
pub/la4
```

from Java to Go. Model your Go solution on the Java solution. Thus, you will have multiple Go packages. Use the patterns we saw in:

```
pub/etc/student
```

To compile your code, execute commands such as:

```
gccgo -g -c customer.go
gccgo -g -c account.go checking.go saving.go
gccgo -g -c bank.go
gccgo -g -c main.go
gccgo -g -o main *.o
```

Then, reimplement the `Accrue` functions, to use goroutines. This is a bit silly, since interest-accrual is too simple to deserve separate threads. In any event, use a channel, so the bank's `Accrue` function can sum the interest, and print the total. Use the patterns we saw in:

```
pub/etc/sleepsort
```

Hints and Advice

- An abstract class/method can be approximated with an interface that declares a function that has no definition.
- Your bank can represent its set of accounts as a map from interface pointers to interface values. An interface value is essentially an object reference:

  ```
  accounts map[*IAccount]IAccount
  ```

  and iterate over them like this:

  ```
  for _,a:=range b.accounts {
      ...
  }
  ```