EXERCISE IN SUBJECT:	**DAT320 Operating Systems**
TITLE:	**Lab 2: Introduction to Go Programming**
DEADLINE:	**Tuesday Sep 14 2014 18:00**
EXPECTED EFFORT:	**10-15 hours**
GRADING:	**Pass/Fail**
SUBMISSION:	**Individually**
REMARKS:	Links in this document are clickable.

University of Stavanger

Faculty of Science and Technology

1 Introduction

In this lab you will be learning the Go programming language, which we will be using throughout the course. Go is a very nice language in many ways. It provides built in primitives to design concurrent programs using lightweight threads, called *goroutines*. In addition, these goroutines can communicate using channels, instead of by means of shared memory which is the most common approach in other languages such as C and Java. Go is also a very small language in the sense that it has very few keywords, and as such it does not take too long to learn the fundamentals of the language.

2 The Go Language

There are many tutorials, books and other documentation on the Go language available on the web. When searching the web, use the word *golang* instead of *go*. The main source of information should preferably be golang.org, which offers extensive documentation. A good book for beginners is Miek Gieben's Learning Go (click here). Another great way to learn Go is to complete the tour linked in the task below.

Task: *Start learning the basics of Go. Complete the tour available on: tour.golang.org. You should do at least the following exercises 23, 35, 40, 55 and 68. You can skip exercises 47, 56-60, and 70. Note that you can change the code inline in the browser and run the code to see the results.*

Later, when you are more familiar with the language, the Go blog has a number of great articles that describe important idioms in the language that may not be obvious; check it out here. And there is of course a number of youtube videos that may also be worthwhile to watch to learn about specific topics, or the language proper.

Go Language Questions

1. Write a loop that repeats exactly n times.
2. What is the value of `ok` in the following example:
```
someMap := make(map[int]string)
someMap[0] = "String"
_, ok := someMap[0]
```
3. Object-oriented programming: How can you define a "class" `Message` with two attributes `Sender` and `Content` in Go?
4. How can a method `CheckForError` be defined on the above `Message` "class", that returns an error? (The function body can be empty.)

3 Writing Go Code

There exist Go support for many editors, among others, vim and Eclipse. See go-lang.cat-v.org/text-editors for others. Note that early versions of the Golang Eclipse plugin was pretty flaky, but this might have improved over the years, so feel free to try it out. Another editor that we have had some experience with is LiteIDE, which is ok, but is a bit tricky to set up the environment to facilitate running your programs from within the IDE. Another great editor with Go support is Sublime Text.

Whichever editor you choose, it is highly recommended that you configure it to use the `goimports` tool. This will reformat your code to follow the Go style, and make sure that all the necessary import statements are inserted (so you don't need to write the import statement when you start using a new package.) The `goimports` tool is compatible with most editors, but may require some configuration.

Note that editors may also be able to run your code within the editor itself, but it may require some configuration. However, using the `go tool` from a terminal window (i.e. the command line) is often times preferred.

4 Installing and Running Go Code

Set up a workspace, try to install and run simple packages, as explained on here (click me). Don't forget to export your workspace as `$GOPATH`. Assuming that you have configured `$GOPATH` correctly, you can run the `go` tool and its subcommands from any directory (most of the time).

Task: Once you have the Go environment set up, you can get the template source files for lab 2 using:

```
go get github.com/uis-dat320-fall2014/labs/lab2
```

This will download some source files for lab 2, which includes a `config` directory and a `main.go` file.

The following step is *optional*, but included to show how to build a package (without a main program). Why is that useful? Well, if you need to use an external package, i.e. a library, that only provide API functions, and no main function, then this is the way to compile and install it. So at this point, you can do the following to install the lab2 `config` package (again this step is optional):

```
go install github.com/uis-dat320-fall2014/labs/lab2/config
```

which will compile and install the files from the `config` directory as a library named `config.a` in a subfolder of the `pkg` directory, which you can inspect as follows:

```
ls -laR $GOPATH/pkg/**/lab2
```

Running `go install` compiles the package named on the command line (as shown above with the `config` package.) However, the `main.go` file declares, `package main` in the first few lines, and also contains a `main()` function. For this case we can do several things, all of which produces runnable code, as outlined below:

```
go install github.com/uis-dat320-fall2014/labs/lab2
```

which will compile and install a binary executable file called `lab2` in `$GOPATH/bin`. Note that `lab2` is the name of the directory in which the main package is found, and is also given as the last element of the path to the `go install` command. To run it simply do:

```
$GOPATH/bin/lab2
```

Or you can also do:

```
cd $GOPATH/src/github.com/uis-dat320-fall2014/labs/lab2
go build .
./lab2
```

which will compile/build a binary executable file called `lab2`, save it in the current directory, and finally run it. Or you can do:

```
cd $GOPATH/src/github.com/uis-dat320-fall2014/labs/lab2
go run main.go
```

All these should result in the same output:

```
{1 hello}
{2 lab}
```

These different approaches to compile/run a Go program may serve different uses depending on what makes sense for different development approaches, and whether you need a binary executable file and whether you need to install it in the default `$GOPATH/bin`.

Note that you may want to include `$GOPATH/bin` in your `$PATH` variable, which will allow you to run the `lab2` binary without prefixing it with a directory.

It is recommended that you use `go install`, since it will produce a binary that you can distribute to other machines of the same architecture. But for quick testing `go run` is also very convenient.

5 Go Exercises

In this exercise you shall extend the `config` package and the `main.go` file.

1. Implement the `parse` function that is missing in the `config` package. Also extend `main.go` to test whether a stored configuration is returned correctly.

2. Storing and retrieving information using text files is error prone and often inefficient. Go includes the `gob` package which allows encoding and decoding Go objects in a binary format. The `config` package includes a method `SaveGob` for storing a configuration in a file named `config.gob`.

   Implement the corresponding `LoadGobConfig()` function and add functionality in `main.go` to test this function. See Section 4.6 in jan.newmarch.name/go/serialisation for an introduction to `gob`. The package documentation for `gob` is also excellent: gob doc.

3. Change the main function to take command line arguments. The following call should create a configuration file with Number=3 and Name=Leander and return "parsed correctly", if the file was created correctly.

   ```
   $GOPATH/bin/lab2 -number=3 -name=Leander
   {3 Leander}
   ```

   You should use the `flag` package in the standard library. See golang.org/pkg/flag for more information on the `flag` package. golang.tumblr.com/post/439732032/flag also has a minimal example.

4. Implement the `Stringer` interface on the `Configuration` struct, and use it for pretty printing configuration objects to produce this output:

   ```
   Name=Leander, Number=3
   ```

6 Deliverable

Complete the `ANSWERS.md` file with answers to all questions. In programming assignments where there are several programming steps, only the last step needs to be included.

Please make sure to run `go fmt yourfile.go` on all source files before handing them in (unless you already use `goimports`, in which case running `go fmt` is unnecessary).