ECGR 4090/5090 Cloud Native Application Architecture
Lab 1: Go Basics

Set up Linux virtual machine on your Windows laptop (for all labs)
Recommend using Ubuntu Linux (22.04 LTS) installed on a Virtualbox VM for all labs.
If your computer does not have sufficient resources (cores, memory) to smoothly run a VM, you could either 1) Use Linux as the only OS, or 2) Dual boot current OS with Linux
You are free to use other Linux distributions or MacOS. You will have to adapt the steps for your OS. No Windows please!

All instructions have been tested on Ubuntu.

Install Ubuntu on Virtual Machine
https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview
Tip: Downloading the Ubuntu image takes a long time depending on your connection speed. Download on a Mosaic machine and save it on a USB drive for the whole team to use. You can share it with other teams as well.

Adjust Screen resolution if needed
https://www.youtube.com/watch?v=w4E1iqsn_wA

Basic Linux commands
https://www.youtube.com/watch?v=IVquJh3DXUA&t=822s

In this lab we will be writing basic Go programs.

Install Go
Follow instructions as provided here to download and install Go in your system
https://golang.org/doc/install

Tip: If you get permission issues, preface the commands by *sudo*
Have ChatGPT handy to get explanations on any topics that you have questions about.

Create directory for GOPATH setting.
mkdir ~/go

For GOPATH environment variable, add the following lines to the ~/.bashrc
export GOPATH=$HOME/go
export PATH=$GOPATH/bin:$PATH

If you need help on configuring paths, bashrc see -
https://linuxhint.com/export-a-path-in-bashrc/

Install your favorite code editor. If you don't have a favorite editor recommend using Visual Studio Code

https://code.visualstudio.com/docs/setup/linux

Configure your favorite editor for Go

For Vim editor (My editor of choice. Use only if you are familiar with Vim)
vim-go plugin
https://github.com/fatih/vim-go-tutorial

For Visual Studio Code
https://dev.to/ko31/how-to-setup-golang-with-vscode-1i4i

Go plugins are also available for Atom and Sublime editors

Writing a simple integer adder program as a package

Go uses a new dependency management system starting from Go 1.11 using Go Modules
A module is a collection of Go packages stored in a file tree with a go.mod file at its root. The go.mod file defines the module's module path, which is also the import path used for the root directory, and its dependency requirements, which are the other modules needed for a successful build.

Create a *labs* directory. We'll keep all the lab assignment code here. Again use AI help if you are not sure how to do this.
Create a *lab1* directory under *labs* directory

Type in following command under the *lab1* directory
go mod init example.com

This creates a go.mod file with a name of the module and the Go version. All dependencies required for building the module will be automatically included here.

We will develop the integer adder program as a Go package

Create a *myadder* directory under *lab1*. All adder package files will be in this directory. It is a good practice to name the directory have the same name as the package (*myadder* in our case)

Cut and paste the following code in a file *add.go* in the *myadder* directory.

```
// Returns the sum of two input integers
package myadder

func Add(x, y int) int {
   return x + y
   //return 42
}
```

Testing is built into Go. The test file is in the same package and is named with "<filename>*_test.go*".
Cut and paste the following code in a file add_test.go in the *myadder* directory.

```
package myadder

import "testing"

func TestAdd(t *testing.T) {
   want := 7
   got := Add(3, 4)
   if want != got {
      t.Errorf("Error in myadder.Add; Want 7, Got %d", got)
   }
}
```

Run the test as follows from command line as follows -
go test

This test should pass.
Now modify *add.go* by commenting out the statement r*eturn x + y*, and uncommenting the statement return 42.

Run the test again. The test should fail.
This completes the first version of the adder package. We can extend the package by adding other functions, and user defined data types. For each component added, you would modify the test to ensure that the component is functionally correct.

For production use, you'd publish the example.com/myadder module from its repository (with a module path that reflected its published location), where Go tools could find it to download it. For now, because you haven't published the module yet to an external repo (example.com

would be replaced by github.com or gitlab.com), you need to adapt *go.mod* so it can find the *example.com/greetings* code on your local file system.

To do that, use the go mod edit command to edit the *example.com/hello* module to redirect Go tools from its module path (where the module isn't) to the local directory (where it is).

From the command prompt in the lab1 directory, run the following command:

$ go mod edit -replace example.com/myadder=./myadder

The command specifies that *example.com/myadder* should be replaced with *./myadder* for the purpose of locating the dependency. After you run the command, the *go.mod* file in the lab directory should include a replace directive, and should be as shown below -

module example.com

go 1.21.6

replace example.com/myadder => ./myadder

Let's now use the Add function from outside the package. Be sure that the test above is passing.

In the lab1 directory, cut and paste the following code to a file named main.go

```
package main

import (
    "fmt"
    "example.com/myadder
)

func main() {
    fmt.Println(myadder.Add(5, 6))
}
```

Notice how the myadder package is referred to with respect to the module path.
Now build and run *main.go* from the command line. Ensure that the test is passing prior to this.

go run main.go

This should print the output of 11.
Note: You can build an executable using go build, and then run it separately.

Congratulations! You have written your first Go program.
In summary, a typical set of steps to write Go code

1. In the development directory, initialize module with go mod init
2. Break your project into a collection of packages
3. Incrementally, develop the code for the package, testing along the way
4. From the module, integrate the packages in main.go for your overall project.

Later we will see how to integrate the development flow with an external code repository such as Github or Gitlab.

To do -
Write a Go program that reads a text file, and outputs the top K occurrences of words in the file.
Words are defined as a set of characters limited by a whitespace.
Write your code in a directory corresponding to the package name (textproc) under lab1
Run *go test* to rest the code.

Hint 1: Use string package for string processing
Hint 2: Use os package for opening a file
Hint 3: Use bufio or os package for file reading
Hint 4: Use a map for counting word occurrences in O(n) time
Hint 5: Helper function for sorting is provided

Please find the following on Canvas - topwords.go (You need to complete this), topwords_test.go, passage