Lab Setup

- Due No Due Date
- Points 0

Step 1: Install Go

You'll implement all the labs in **Go** (http://www.golang.org/) . The Go web site contains lots of tutorial information. You should use Go 1.15 or any later version. You can check your Go version by running `go version`.

We recommend that you work on the labs on your own machine, so you can use the tools, text editors, etc. that you are already familiar with. Many editors have plug-ins for Go. We are happy to provide support over **MS Teams** (https://teams.microsoft.com/l/channel/19%3A440bb6d5ec974fa7871e10c5ffd3f318%40thread.tacv2/Lab%202?groupId=ebbe7cd7-cfc6-4d05-b84f-6164f484dc15&tenantId=5217e0e7-539d-4563-b1bf-7c6dcf074f91) and in office hours for helping you set up tooling. As a fallback, you can work on the labs on CADE, but we do not recommend this option.

Once you have a working Go install, you'll have to decide what other support tools you'd like to use. For example, VS Code has excellent Go support and will automatically suggest and install extensions for things like an interactive visual debugger and unit test integration.

macOS

You can use **Homebrew** (https://brew.sh/) to install Go. After installing Homebrew, run `brew install go`.

Linux

Depending on your Linux distribution, you might be able to get an up-to-date version of Go from the package repository, e.g. by running `apt install golang`. Otherwise, you can manually install a binary from Go's website. First, make sure that you're running a 64-bit kernel (`uname -a` should mention "x86_64 GNU/Linux"), and then run:

```
$ wget -qO- https://go.dev/dl/go1.18.4.linux-amd64.tar.gz | sudo tar xz -C /usr/local
```

You'll need to make sure `/usr/local/go/bin` is on your PATH. You can do this by adding `export PATH=$PATH:/usr/local/go/bin` to your shell's init file (commonly this is one of `.bashrc`, `.bash_profile` or `.zshrc`).

Windows

The MIT labs don't recommend the use of Windows for these labs. You may try WSL2, which they haven't tried but sounds pretty likely to work using the Linux instructions (based on Ryan's other WSL2 experiences). Alternatively, you can use the CADE machines (see below).

Using CADE

CADE (https://www.cade.utah.edu) manages two clusters that you can use to do your development and testing for all of the class projects. You are free to use other machines and environments. I only recommend using CADE if doing development locally isn't practical for you.

Check with **CADE (https://www.cade.utah.edu)** if you need to setup an account; they have a **special form you fill out to create your account (https://usertools.eng.utah.edu/create_account/index.php)** if you don't have one already.

CADE machines all share your home directory, so you needn't log in to the same machine each time to continue working.

Please try these steps, if you get stuck please post to the **Lab 2 channel on MS Teams (https://teams.microsoft.com/l/channel/19%3A440bb6d5ec974fa7871e10c5ffd3f318%40thread.tacv2/Lab%202?groupId=ebbe7cd7-cfc6-4d05-b84f-6164f484dc15&tenantId=5217e0e7-539d-4563-b1bf-7c6dcf074f91)** or send a Canvas message to the TA and/or professor or come to office hours.

Setting up and installing software

Shell to a CADE machine

After you have an account choose a machine at random from from the lab1- set of CADE machines (that is, lab1-1.eng.utah.edu through lab1-40.eng.utah.edu).

```
ssh lab1-10.eng.utah.edu
```

As of 2023, accessing the CADE machines from off campus requires manual authentication each time, making it difficult to use certain software (VS Code's remoting functionality). You can fix this using the VPN to connect to the campus network, or if you have ssh access to another machine on the campus network you can use some fancy forwarding to get this to work -- ask in Teams if this is something you want to set up yourself.

Run bash

CADE user accounts have `tcsh` set as their default shell. Each time you login first run `bash` before anything else. All instructions, examples, and scripts from this class assume you are using bash as your shell.

You'll need to do this each time unless you **reset your default shell (http://www.cade.utah.edu/faqs/can-i-change-my-default-shell-from-tcsh/)** (which I'd recommend). (You may want to look up how to change the bash command prompt online; for example adding `export PS1='[\u@\h \W]\$ '` to your `~/.bash_profile` will give you a more informative shell prompt the next time you log in.)

This step is important. If you don't reset your shell, other things will mysteriously break as you try to work through the labs.

Install Go in your home directory

go is the language we'll be using in our labs. The compiler and toolchain comes in a pre-compiled tarball. It just needs to be downloaded and untared and it's pretty much ready to go. Since CADE machines all share home directories, installing it once in your home will work across all the lab machines.

```
$ wget https://go.dev/dl/go1.18.4.linux-amd64.tar.gz
$ tar zxvf go1.18.4.linux-amd64.tar.gz
$ echo 'export GOROOT=$HOME/go' >> ~/.bashrc
$ echo 'export PATH=$GOROOT/bin:$PATH' >> ~/.bashrc
$ source ~/.bashrc
```

You may also like to **read more on how to install** (https://golang.org/doc/install) go wherever you like.

Test Go

Use your favorite editor (`emacs`, `vim`, etc.) to create `test.go` with the following contents:

```
package main

import "fmt"

func main() {
  fmt.Printf("hello, world\n")
}
```

Then run

```
$ go run test.go
hello, world
```

If you see "hello, world", then you're ready to get setup with the lab code.

If you are a vim user you might consider using **vim-go** (https://github.com/fatih/vim-go) . Other IDEs have similar helpful sets of extensions for formatting/checking code, etc.

If you are comfortable using vscode then you might consider doing **Remote Development Over SSH,** (https://code.visualstudio.com/docs/remote/ssh) where you install vscode on your local machine and connect it to CADE machine for development. You can also add **Go extension** (https://marketplace.visualstudio.com/items?itemName=golang.Go) for additional go related features. vscode also includes support for interactive debugging of Go code, which can be very helpful in debugging the assignments.

Step 2: Getting the Lab Skeleton Code

Clone our course lab skeleton code; this same skeleton code will be used for all of the lab assignments.

```
$ git clone https://github.com/rstutsman/cs6450-labs.git
Cloning into 'cs6450-labs'...
remote: Enumerating objects: 122, done.
remote: Counting objects: 100% (122/122), done.
remote: Compressing objects: 100% (80/80), done.
remote: Total 122 (delta 38), reused 122 (delta 38), pack-reused 0
Receiving objects: 100% (122/122), 1.26 MiB | 3.18 MiB/s, done.
Resolving deltas: 100% (38/38), done.

$ ls
cs6450-labs
```

If that all works you should now have a new directory called `cs6450-labs` with all of the labs' skeleton code inside. You can do your development inside that directory. I recommend making regular commits within your repo and using different branches for each assignment, but you can structure things however you like. Each assignment includes specific instructions on how to upload your code for grading.

DO NOT share this code online in any public repo either with or without changes you've made. If you do want to place a copy of it on e.g. Github it is fine so long as the repo is marked private. Exposing your code publicly is academic misconduct.

Make sure you have read and fully understand the Academic Misconduct Policy on the syllabus for this course since it includes special restrictions on sharing and use of Generative AI tools with the code.

Step 3: Developing Go

You'll want to walk through the **Go Tour** (http://tour.golang.org/welcome/1) after you've got a working installation. The tour is interactive - so it'll give you an easy way to play with the language and get a sense of how it differs from C, C++, Java, etc.

Effective Go (https://golang.org/doc/effective_go.html) is an optional but interesting read that will undoubtedly improve your Go code - it details nearly all of the more nuanced aspects of the language.

It may be helpful to read **How to Write Go Code** (https://golang.org/doc/code.html), which outlines the details of the `go` tool, though we'll only use a subset of its capabilities for the labs. Go has a stylized way to run, compile, and test code - this document gives the full details of how it works.