ECE 264: Advanced C Programming

Lecture Notes 6/10/19

1 Topics

1. We discussed the syllabus
2. GitHub (live demo)
   (a) Initializing a repository: cloning a git repo
      i. Using HTTPS on ECN machines
      ii. Using SSH (including generating SSH keypair)
   (b) Adding content: how to add and commit
   (c) Saving content on remote: how to push
   (d) Making releases: how to tag
3. Makefiles
   (a) Format
   (b) Macros
4. Selection sort (brief introduction)

2 Version Control

This class uses Git for version control. Git is a distributed version control system. That means there are two repositories: local and remote. When you commit changes, only the local repository is changed. This makes commits fast and independent of network connections. If your computer is damaged, you still lose the local repository. To change the remote repository, you need to push the changes. If your computer is damaged, you can retrieve the code from the remote repository. Please push your code to GitHub often. Not only does that prevent you from losing any code if you accidentally delete anything, it helps us help you debug, by giving us access to your latest code.

Please read the book about how to use version control: https://git-scm.com/book/en/v2

We did a live demo showing the following steps:

1. Show HTTPS clone on ecegrid machine
2. Show SSH setup and clone on ecegrid machine
3. Show git status, git add, git commit, git push
4. Show git tag

Setting up a GitHub Account create a Github account (if you do not already have one). This is the account you should use to create and submit all of your assignments this semester. Also let us know your account name by filling in the Google Form provided in BlackBoard and Piazza.

Setting up SSH key with GitHub Set up a public SSH key in your GitHub account (if you haven't already). To do this, first generate a new ssh key:

```
    > ssh-keygen
```

Hit enter three times (to accept the default location, then to set and confirm an empty passphrase). This will create two files: /.ssh/id_rsa (your private key) and /.ssh/id_rsa.pub (your public key) Then print out your public key:

```
    > cat  /.ssh/id_rsa.pub
```

And copy it to the clipboard. Then follow steps at: https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account

3 Makefile

Makefiles let you define complicated sets of commands to build your projects.

Makefiles consist of a series of rules:

```
[target] :  [dependences]
[TAB]   command 1
[TAB]   command 2
...
```

A rule *target* is the name of the rule. The *dependences* are the files the rule depends on. The *commands* are what to do when the rule is "fired". Note: there must be a tab before each command.

A rule is fired in one of two ways: (i) it is directly invoked (by calling "make [target]") or (ii) it is invoked by another rule that is fired.

When a rule is fired, it goes through the following process:

1. If a *dependence* has a rule in the Makefile, fire that rule (using this same process)
2. Once all dependences have been fired, check to see if *target* is "out of date": interpret *target* as a filename, and see if the timestamp on the file is older than the time stamp on any of its dependences. If it is, the target is "out of date". If there is no file named *target*, the target is always assumed to be out of date. If there are no dependences and *target* exists, target is assumed to be up to date.
3. If the target is out of date, execute the list of commands

You can use Makefiles to orchestrate complicated build processes.
If you type "make" without a target, make will fire the first rule in the Makefile.
We usually define a target called "clean" whose job it is to clean up any intermediate files generated during the build process. This can also be used to remove all generated targets to force recompiling everything.

Makefiles also let you define macros to reuse the same commands over and over. For example, we can define GCC as a macro that invokes gcc the way we want:

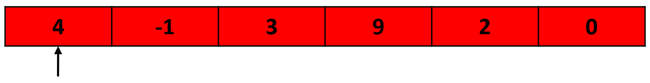
```
DEBUG = -DDEBUG
CFLAGS = CFLAGS = -std=c99 -g -Wall -Wshadow --pedantic -Wvla -Werror
GCC = gcc $(CFLAGS) $(DEBUG)
```

Note that we use $(MACRO_NAME) to insert the macro into other places, including commands. Makefiles can get much more complicated than this, but their full power is beyond the scope of this course.

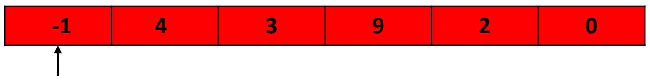
4 Selection Sort

Readings: Forouzan and Gilberg, pp 491-493

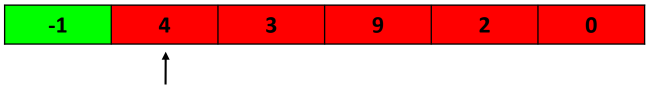
Selection sort is one particular sorting algorithm that sorts an array using the following procedure: Divide the array up into two pieces (we'll call them "sorted" and "rest"). *Sorted* is the portion of the array that is already sorted. *Rest* is the rest of the array. One thing that is always true (an invariant) is that all the elements in *sorted* are smaller than any of the elements in *rest*. Selection sort works by slowly growing the sorted side of the array and shrinking the rest of the array. Think of this as having a cursor. When the sorting starts, we don't know if any of the array is sorted, so our cursor starts out pointing to the first element of the array. Everything to the left of the cursor (colored green, in this case nothing!) is sorted:



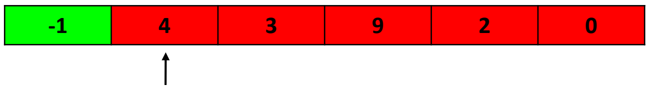
Everything from the cursor to the right (colored red) is the rest of the array. We then scan through the rest of the array to find the smallest element, and swap it with the element at the cursor (this might be the element itself!):



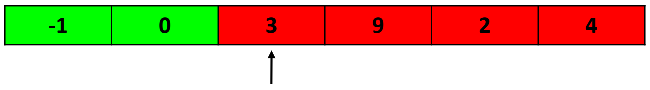
Because we just moved the smallest element to the cursor, we can now move the cursor up one:



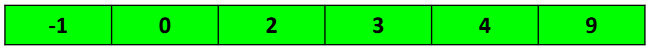
Note that our rules still hold: everything to the left of the cursor is sorted, and everything from the cursor right is larger than anything in the sorted part of the array. Now we repeat the process, finding the smallest element in the rest of the array and swapping it with the cursor:



And we can now move the cursor to the right again, restoring our properties. The sorted part of the list is sorted, and the rest of the list is larger than anything in the sorted part of the list:



Note that each time we repeat this process, the cursor moves one to the right. The sorted list gets longer, and the rest of the list gets shorter. In this manner, we eventually sort the list:



We will use pseudocode for most of our code examples in class. This lets us quickly explain the structure of an algorithm without worrying about nitty-gritty details of correct C syntax. It also means that we can describe an algorithm without giving you code that you can just copy for an assignment!