# ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 32

Vince Weaver

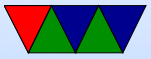
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 April 2026

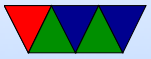
# Announcements

- HW#7 is due
- Update on Lab – will have last checkoff next week
- Title II was averted
- Student Evals (if you only do a few, the main section one is the most important, more than the lab or recitations)
- There is a final, Wednesday 1:30pm during finals week. More on that next week.



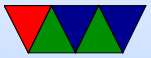
# Writing Good Code

- How do you define good code?
- Does the job it is supposed to
- Does it with high performance?
- Is it maintainable? (this might be the most important long-term)



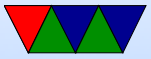
# Maintainable Code

- Well structured
- Good variable names
- Broken up into smaller functions if possible
  Avoid deeply nested control flow
- Also can break up into multiple .c files
- Good comments
- Good git commit messages



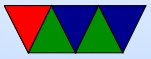
# Testable Code

- Test suite (test code you run to make sure nothing breaks)
- Should test likely things, but also check unusual or corner cases
- When do you run?
  - After change
  - Every evening
  - Just before release?

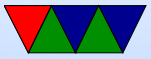


# SCM – Source Code Management / Version Control

- In old days you might have one C code file you worked on
- What happens if you:
  - Over-wrote / erased by accident?
  - Decided a change wasn't good and want to roll back?
  - Wanted to know when and why a change was made?
  - Found a bug and wanted to know when it was introduced?

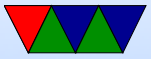


- Wanted to collaborate on code with someone else?



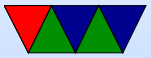
# Before SCM

- Work on code
- Occasionally back up, or worse have like lots of files with names like code.c , code_good.c, code_good2.c, etc
- To release the code you might tar/zip it up and put it on a website with a version number attached



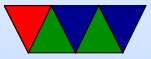
# Collaborating Before SCM

- If people wanted to contribute they would download, work on the code, then e-mail you "patches" made with the `diff` tool
- Diffs showed the difference between your code and theirs.
- Use the `patch` tool to apply these changes
- Linux operated this way for a long time



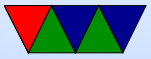
# SCM – Tools

- Would it be better if you could have some sort of automated way to track code, changes, etc?
- First SCM tools let you do this, SCCS, CVS. Various commercial offerings too
- You'd "check in" your code with a message
- You'd work on the code, and when you made a useful change, check in that too
- You can view the changes made over time, roll back to previous changes, have messages/commit messages



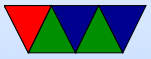
# SCM – Tools More Modern

- The tools had limitations. CVS (concurrent version system), the most common free one, wasn't great
- For a while more modern replacements: SVN (subversion), Mercurial (HG), seemed like they'd take over



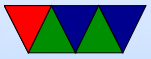
# SCM – Linux

- One of largest open source projects Linux, Linus overwhelmed. Push for him to use SCM but he didn't like any
- Larry McVoy designed bitkeeper, proprietary SCM, but let Linux use if for "free" (but closed source). Linus liked it and used it
- Other Linux developers didn't like closed source, especially their changes being locked away in it
- Andrew Tridgell (samba developer) started reverse



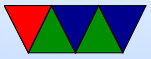
engineering it, McVoy got mad and threatened to not let Linux use it

- Linus took a break, and over the course of a few weeks invented his own SCM, “git”



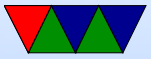
# SCM - git

- Most popular SCM these days
- Don't look into how it works, becomes infinitely complex with diagrams and hashes
- Each change you check in gets a SHA cryptographic hash, long chain of numbers (turns out the hash they picked was weak, trying to replace it)
- You can use these to track all changes
- Can have separate branches, tags, other things



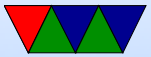
# git – websites

- Git is decentralized, unlike some others
- git tree includes all code and version info. Takes up disk space, but it means everyone has full backup
- For collaboration websites like github sprung up, they can act as a central point for development. Add extra things like build farms, bug trackers, etc.
- Microsoft bought it
- Note, you can use git standalone, you don't have to use github

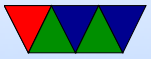


# git – common commands

- `git clone` – check out a repository
- `git log` – show recent checkins
  `git log -p` to show code change for each checkin
- `git add` – add all changes to file for next checkin
- `git diff` – see changes
- `git commit` – create a commit with all changes added recently
- `git push` – push changes into repository
- `git pull` – pull down any new changes

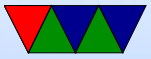


- `git blame` – show who made changes to each line of code
- there are many, many, many more



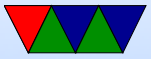
# Debugging

- You wrote the best code ever
- But it doesn't work
- How do you debug this?
- It turns out debugging is much harder than writing code



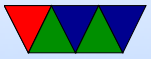
# What if the Code was Previously Working

- Try to find what change broke things
- If using git, can use “git bisect” to do a binary search. Check versions between known working and broken and find out what broke it



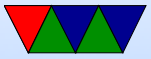
# What if the Code just doesn't do what I want

- Traditionally there are three ways to go about this
  - Stare at the code until you figure it out
  - Add debugging code that prints/logs what is going on and see if you can find a discrepancy
  - Use a debugger



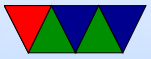
# Staring/Thinking method

- Not always the most effective
- Easy to miss small bugs because your brain expects the right thing
- Having a friend look can help
- Explaining to a duck can help



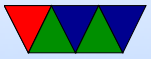
# Printing/Logging Method

- If something is supposed to happen, but doesn't, try to track down why it isn't
- If a value needs to be set, verify it is. Can use `printf()`



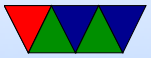
# Printing/Logging Method

- If you aren't reaching code you expect, can also sprinkle printf()s along the way and you can see which ones get called to verify control flow
- Sometimes this is a lot of output to sort through



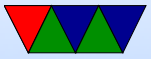
# What if printf() not possible

- There are times you aren't able to easily printf()
- embedded systems, low-level OS
- (I've had to resort to blinking LEDs at times)



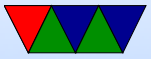
# Debugger

- Powerful tool that lets you see exactly what code is doing at machine code-level
- Can step through code line-by-line, see individual memory bytes, register contents, stack, etc
- Set breakpoints (have code execution stop at exact location
- Set watchpoints on memory locations (can't figure out how `x` gets value 25? Can tell debugger to stop each time `x` gets written to



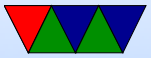
# Debugger Downsides

- Harder (though not impossible) to use with code running on embedded system like the Pico. Device has to support JTAG or similar
- The most common Linux debugger `gdb` is a horrible command line program that's obscure to use.
  Not sure why a better alternative hasn't taken off



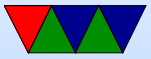
# Using gdb

- Be sure it is installed
- Run `gdb ./program`
- Type `run` for it to run (also include command line)
- You will get better results if you compile your programs with debug info `gcc -g`. This will make your programs bigger (you can later strip off the debug info with the `strip` command



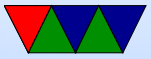
# Using gdb example

- `bt` backtrace
- `info regis` gives register,
- `disassem` disassembles, etc.
- `cont` continue running
- `break` sets breakpoint



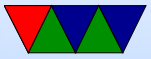
# Using valgrind example

- Install valgrind
- Run `valgrind your_program`
- Will give you a summary of memory leaks and such
- Best at dynamically allocated memory (malloc/calloc, etc)
- Bunch of other tools it provides, also good at pthread race conditions



# Is it a Compiler/Operating System/Security Bug

- It can be, but the odds in a class like this are low
- If it is though, it can be fun to track down and report it so it can get fixed



# Code is Fixed, Now what

- Maybe update comments if necessary
- If you have a test-suite, write a test that triggers the bug and that way it will be caught it you accidentally re-introduce it