ECE 471 – Embedded Systems
Lecture 28

Vince Weaver

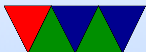
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

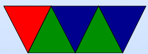
15 November 2024

Announcements

- HW#9: the time question not aimed at anyone in particular at UMaine, it dates back to some issues I had when I was a TA as a grad student
- Remember to pick up project parts sooner rather than later
- No in-person class Monday. I'll post the homework review notes and try to post a video of some sort of me going over it
- Wednesday the midterm will be in-class, will be proctored

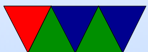


by someone other than me

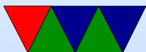


Midterm #2 Preview

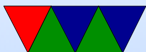
- Can have 1 page of notes like last time
- Primarily material since the last midterm
- Booting on the Pi
  - What a bootloader does
  - Why boot firmware and early filesystems need to be simple
  - Why Pi is unusual
- Real Time
  - Definitions



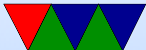
- Is this hard, soft, firm
- i2c/SPI/1-wire
  - Know the tradeoffs between i2c, SPI, 1-wire
  - Be able to follow the C code for them
- Security
  - Buffer overrun, why it is bad
- Coding Practices
  - Be aware of the case studies we suggested
  - Know of some of the recommended ways to write safer C code



Finish some examples from last time

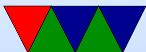


How Can You Avoid Bad/Buggy Code?



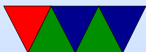
Code Safety Standards

- Avionics: DO-178C (1992 for B)
- Industrial: IEC 61508 (1998)
- Railway: CENELEC EN 50128 (2001)
- Nuclear: IEC 61513 (2001)
- Medical: IEC 62304 (2006)
- Automotive: ISO 26262 (2011)



Code Safety Standards

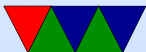
- Is it easy to get a hold of copies of these?



Automotive ISO 26262

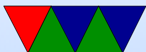
What is a document like this like?

- Vocab and definitions
- Management
- Safety Life Cycle
- Supporting processes
- Safety Analysis
- Risk Strategy
- Severity
  - S0 – No injuries

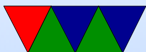


- S1 – No injuries
- S2 – Severe injuries
- S3 – Not survive-able

- Exposure
  - E0 – Unlikely to Happen
  - ...
  - E4 – High probability
- Controllability
  - C0 – Controllable
  - ...
  - C3 – Uncontrollable

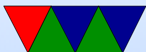


- Look up those in a matrix so you know how to assess risk, know how important to fix, know what resources to apply



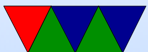
Aviation

- DO-178B / DO-178C
- Software Considerations in Airborne Systems and Equipment Certification
  - Catastrophic: fatalities, loss of plane
  - Hazardous: negative safety, serious/fatal injuries
  - Major: reduce safety, inconvenience or minor injuries
  - Minor: slightly reduce safety, mild inconvenience
  - No Effect: no safety or workload impact



Medical Response

- IEC 62304 – medical device software – software lifecycle
  - Quality management system – establish the requirements needed for such a device, then design methods to be sure it meets these
  - Avoid reusing software of unknown pedigree (don't just cut and paste from stackoverflow)
  - Risk management – determining what all the risks involved are, then determine ways to avoid or minimize them



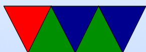
◦ Software safety classification

Class A: no injury possible

Class B: Nonserious injury possible

Class C: serious injury or death possible

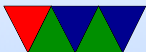
Software sorted into these areas. Class A do not require the same precautions as the others.



Other notes

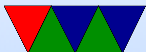
- Top down vs Bottom up Design
  Spec out whole thing and how they work first
  Start with core part and just keep adding to it until it works

- Requirements/Specifications?



Writing Good (Embedded) C Code

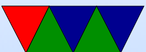
- Various books
- Comment your code!
- Strict, common code formatting (indentation)
- More exact variable types (int32_t not int) Size can vary on machine, and on operating system
- Subset to avoid undefined behavior
  - In C standard some relatively common behavior can be "undefined"
  - Signed integer overflow, shifting left by 32, order of



evaluation of command line paramaters

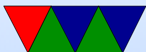
◦ `printf("%d %d\n", ++i,i++);` different on different machines

- Tool that enforces the coding standards
- Good to write safe code even if it isn't meant for a safe application. Why? Good practice. Also who knows who or when your code might be copied into another project.



MISRA

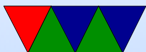
- Next time look at links here: `https://news.ycombinator.com/item?id=38674158`
- **MISRA: Guidelines for the Use of the C Language in Critical Systems**
- Motor Industry Software Reliability Association
- Guidelines: Mandatory, Required, Advisory
- Some sample guidelines
  - Avoid compiler differences int (16 or 32 bit?) `int32_t`
  - Avoid using functions that can fail (malloc()) allocate



memory at beginning of program not throughout

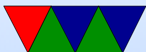
- ◦ Maintainable code, comments, coding style (see below)

- Compliance
  - ◦ All mandatory rules must be met
  - ◦ All required rules must have formal deviation
- Deviation
  - ◦ Must make a format explanation for why deviation is necessary
  - ◦ Prove you've thought about the issue
- MISRA 2012 has 143 rules, 16 directives



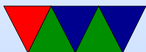
- NOTE: YOU CAN STILL WRITE BAD CODE EVEN WHEN FOLLOWING THIS

  It just makes it easier to write good maintainable code.



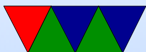
C Style

- What can C look like?
  IOCCC (International Obfuscated C Code Competition)
- Variable style, `CamelCase`, `under_score`, Hungarian Notation (arru8NumberList)
- Indentation (tabs vs spaces)
- Curly braces on same or next line
- Comment style
- Auto-generated documentation from comments



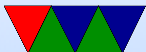
Good Test Practices

- Unit testing
- Test Driven Development – tests written before the code happens, needs to pass the tests before done
- Fuzzing
- Device Hardening?



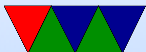
Good Documentation Practices

- Comment your code
- Write documentation! Make sure it matches code! There are some tools that can auto-generate documentation from special code comments
- Use source control (git, subversion, mercurial)
- Use good commit messages in your source control

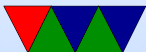


Space Shuttle Design

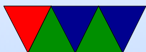
- https://www.nasa.gov/mission_pages/shuttle/flyout/flyfeature_shuttlecomputers.html
- Issues normal embedded systems don't have: Vibration at liftoff, Radiation in Space
- If computer stopped for more than 120ms, shuttle could crash
- "Modern" update in 1991: 1MB Ram, 1.4MIPS. Earlier was 416k and 1/3 as fast and twice as big
- Change to code, 9 months testing in simulator, 6 months more extensive testing



- 24 years w/o in-orbit SW problem needing patches
- 12 year stretch only 3 SW bugs found
- 400k lines of code
- HAL/S high-order assembly language (high-level language similar to PL/I)
- PASS software – runs tasks. Too big to fit in memory at once
- BFS – backup flight software. Bare minimum to takeoff, stay in orbit, safely land, fits in memory, monitors pASS during takeoff/landing Written by completely different team.



- 28 months to develop new version
- IBM
- Extensive verification. One internal pass, one external
- 4 computers running PASS, one running BFS
- Single failure mission can continue; still land with two failures
- 4 computers in lock-step, vote, defective one kicked out



SpaceX Falcon 9

- Linux – on dual core x86 systems
- Three each, vote
- Flight software in C/C++
- Dragon displays in Chromium+JS