ECE 471 – Embedded Systems
Lecture 26

Vince Weaver

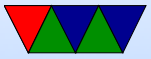
https://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

7 November 2025

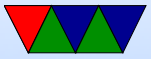
Announcements

- HW#8 is due
- Project topics are due
- Midterm #2 on Wednesday November 19th
- HW#9 will be posted, you can have two weeks as it's a bit harder



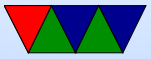
Hand back and go over Midterm

Average grade was an 88%



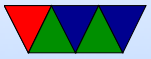
HW#9 – Summary

- Use a temperature probe (either SPI or 1-wire) and output the result to the i2c display
  - Re-use i2c display code from earlier homework
  - Re-use temp code (either TMP36 or the 1-wire)
  - Display the temperature on display
- When done can turn back in parts (assuming you aren't using them for the project)



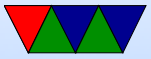
HW#9 Notes – Modular Code

- In previous homeworks we put everything in one C file
- This isn't really practical for large projects
- By splitting things up into smaller files you can have some benefits:
  - Easier to organize/find code
  - Can re-use code easier
  - Less chance of merge conflicts when multiple people working on project in git
  - Can take common code and make libraries



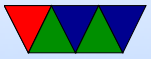
HW#9 – Writing Modular Code

- In C you can compile each C file into its own object file, link together at end
- API defined in a header .h file
- For example in the homework, we could put temperature read code into its own file with a `double get_temperature(void)` interface
- For other C files to see this, you need to export the definition. Usually this is done by putting the advance definition `double get_temperature(void);` in a .h



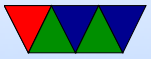
header file and then including it in the other files

- Note: don't put full C functions in header files. I know this is a C++ thing but it's usually frowned upon when programming in C
- Each file does not need a `main()` function, you only need one per combined program.



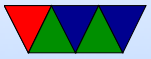
HW#9 – Building Modular Code

- To link the various .o files together involves the "linker". However it's easier to just let gcc do it (gcc knows how to run the linker for you) `gcc -o display_temp display.o temperature.o`
- The linker merges the .o files into one big executable, and makes sure the placeholders to functions/variables in all of the files get the right addresses/pointers to where things live in the finished executable.
- How do you make sure when you change one C file that



everything that uses it is also rebuilt? A well-crafted Makefile will have all these dependencies in place and will rebuild everything properly.

- What if you want to make an official library? Static libraries are .a, dynamic .so. It's fairly easy to do this, just a few extra command line tools like `ar` or maybe even just using `-shared` to gcc



HW#9 – Converting Floating Point to Digits

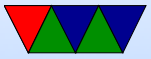
- Use sprintf()

```
char string[128];
double temperature;
sprintf(string,"%.1lf",temperature);
/* Now string[0] has first digit, string[1] second, etc */
```

- Use division/modulus

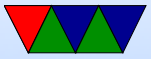
```
double temperature=23.4;
int hundreds,tens, ones,remainder;

hundreds=temperature/100;
remainder=temperature%100;
tens=remainder/10;
ones=remainder%10;
```



HW#9 – Specification (4 cases)

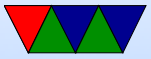
- Temperatures from 0 to 99.9 degrees, inclusive. $0.0 \leq temp \leq 99.9$ These should be displayed as two digits, a decimal point, another digit, and then a degree symbol (which is just a crude circle made of the top 4 segments on the display). For example: "24.5°"
  Leading zeros should be suppressed (i.e. display "2.4°" not "02.4°", 0 should be "0.0°")
- Temperatures between -99.9 and 0 degrees. $-99.9 \leq temp < 0$ These should display a minus sign and then



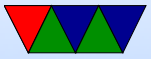
two digits of temperature, then the degree symbol. For example: "-25°"

For temperatures between 0 and -9.9 be sure to print two digits of result (with decimal point). For example, "-2.5°"

- Temperatures between 100 and 999 degrees. $100 \leq temp \leq 999$ should print three digits of temperature, then the degree symbol. For example: "245°"
- Invalid temperatures that won't fit the display (and errors reading the thermometer) should be reported (via the display) in a method that isn't a valid temperature. It is



your choice how to indicate this.



HW#9 – Writing Good Testcases

- Once you have written your nice modular code, how can you test it?
- Need to write some test cases that test a wide range of behaviors
- In the homework I have you think up some test cases