ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 6

Vince Weaver

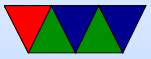
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

4 February 2026

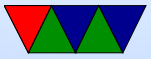
Announcements

- Homeworks, working on getting that set up
- Accessability office looking for note-takers, they said they e-mailed and no one volunteered so they asked me to mention it



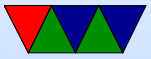
Lab – First Finish up Lab#1

- Finish getting WSL and VS Code set up
- Always bring your laptop and parts to lab



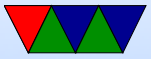
Lab#2 Notes

- Once again thanks for Tuesday lab for being first
  this one was more last minute than it should have been



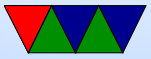
Lab#2 Topics

- Test out some of the numbering system things we've discussed
  - Printing Hex, Decimal, and Binary Numbers
  - Seeing some weird two's complement behavior
- Practice programming in C at the command line with a text editor
- I added in something "fun" at the end, change color of the text you print



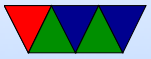
Lab#2 Example Run

- I'll run through what it looks like on the Pi
- Two screens, one with editor, one with compiler
  This keeps you from having to exit and restart editor over and over
- Add code to do number conversion in C
- Show off Linux/command line tricks, like up-arrow for history, tab-completion, cut-and-paste in nano



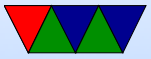
Lab#2 – Quiz/Worksheet

- I have 15 sets of values to convert
- You enter these into a Brightspace worksheet which will let you know if they are right or not
- The idea is you do these conversions by adding a line to your C code to do it for you
- You must do the first 3 questions using C. If you'd rather do the rest by hand that's fine
- At end take a few screenshots



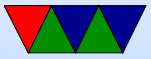
Lab#2 – Problems found, it's never easy

- On MacOS the C library doesn't seem to support `printf("%b");` even though it's a C23 feature
  If you're on mac you might have to print hex and manually convert to binary (that's relatively straightforward)
- Brightspace fill-in-the-blank grading has trouble with negative numbers. Minus sign (ASCII 0x2D) and hyphen (Unicode character U+2010) can look the same, but don't match so it was marking them wrong



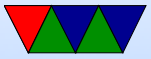
Lab#2 – Manipulating Files on Windows/WSL/Linux

- Finding files on Windows/WSL, under `/mnt/c/Users/USERNAME/Do`
- If you have OneDrive enabled might be under `/mnt/c/Users/USERNAME/Documents/ece177/....`
- Note on windows vs Linux the file separators are flipped, long ago historical reasons
- On Linux/MacOS can get to your home directory quickly with ~



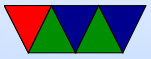
Angband

- Show off a game that was originally done entirely in ASCII art / colors
- Written in C
- A “roguelike”
- Tolkien themed
- Hard to beat due to permadeath



printf()

- Function provided by C library that prints to standard output (stdout)
- Declared in `stdio.h`
- The first argument is ALWAYS a string

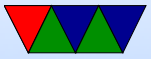


Aside: what is a string?

- A string is a list of characters that are grouped together

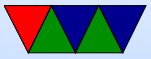
```
'a'     // is a one byte character
"Hello" // is a 6-byte string made up of 5 characters an
        // "NUL" terminator (0 at the end)
        // When in double quotes the NUL is
        // automatically put in for you
```

- Strings cause a lot of trouble in C, we'll come back to them later



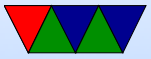
Format String

- The first argument is acalled the format string
- `printf()` outputs the characters one by one until it encounters any special characters:
  - Escape characters (we mentioned these with constants) that start with a backslash (i.e. `\n`)
  - Formatting specifiers which start with a percent sign and tells printf to substitute a value



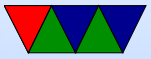
printf parameters

- After the format string, pass in parameters as necessary
- Need one value for each specifier
- Parameters are used from left to right



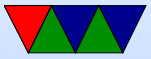
printf specifiers – integers / characters

`%d` or `%i`	signed decimal integer
`%x` or `%X`	hexadecimal (lower/uppercase)
`%o`	octal
`%b`	binary (C23 and later)
`%u`	unsigned decimal
`%c`	raw character (print as ASCII)
`%s`	string
`%p`	pointer



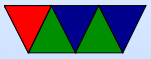
printf specifiers – floating point

`%f`	float
`%lf`	double
`%e` or `%E`	exponent notation
`%g` or `%G`	automatic f or e



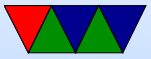
printf specifiers – other

%%	if you want to print percent
%n	output number of characters converted (don't use)



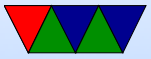
printf length modifiers

- Placed after the % but before format
- h – short integer (16-bit) `"%hd"`
- hh – 8-bit value `"%hhd"`
- l – long integer `"%ld"`
- L – long double



printf fancy formatting

- Field width – `"%4d"` prints 4 digits
- Precision – `"%.3f"` floating point, 3 digits after decimal
- Padding, leading 0s – `"%04d"`
- Justification, right or left (default right) – `"%-d"`
- Sign – `"%+d"` (default only negative get sign)
- Alternate form – `"%#"` various things, if hex or octal will put 0x or 0 in front

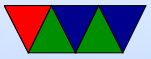


printf – promotion

- This is complicated because it takes a varying number of arguments (vardiac function)
- Generally small values like char/short promoted to int before passing in
- printf() can't tell the type you actually pass in, and will just treat it like you tell it. so

```
float f;
printf("Print an int: %d\n",f);
```

  Isn't necessary smart enough to cast to an integer for you, it will take the bits in the float and try to print



them

- Why? Remember, printf() is a library function, not a part of the C language
- Recent compilers are smart enough to know about printf specifiers and warn you