ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 1

Vince Weaver

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

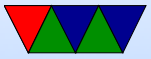
vincent.weaver@maine.edu

21 January 2026

Welcome to ECE177!

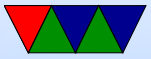
We're going to learn all about C!

`https://web.eece.maine.edu/~vweaver/classes/ece177_2026s/`



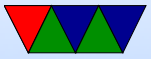
Announcements

- No Labs this week!
- Will start taking attendance next week!



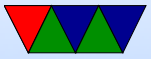
Syllabus – Instructor Info

- I'm Professor Weaver
- First time teaching this class so please bear with me
- Go over syllabus (can find it on website)
- QR-Code: Should you trust it?



Syllabus – Academic Honesty

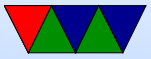
- This has been a problem in the past!
- Do not copy code from other students, either current or from previous years.
- Asking help from the professor/TA is fine
- General discussion with classmates is fine
- Even having someone look over your code to help find a problem is fine
- Do not share your code, even if the person requesting claims they will just use it as a reference. In



my experience it's too tempting and the person will "accidentally" submit it as their own.

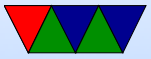
- Do not share code even if it's after the submission deadline as students will submit late work.
- Just don't copy someone else's code and submit it as your own

  This includes cut-and-paste or retyping
- Also don't copy code off the internet (again, looking for advice online is fine, but copying code directly is not)
- Don't use AI tools that do the homework for you! (Like Microsoft/Github Co-pilot/ChatGPT)



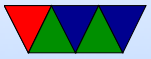
Why not AI?

- You'll note that I'm not a huge fan of AI
- Makes me unusual as it's the current fad
- You're here to get a solid C background for later classes/life
- AI can be subtly wrong, and you can only catch it if you actually know what's going on
- I want to be helping you learn to code, not some AI
- Who knows what happens once the AI bubble collapses, or if/when they start charging a lot of money to use AI



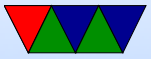
Programming

- Has anyone programmed before?
- What is programming?
- We want to tell a computer what to do.
- It shouldn't be an unfathomable magic box, but it's a tool that we can understand and control
- I'm a computer engineer and like to program, I know it can be tedious for those who don't



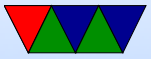
Programming Languages

- What languages have you used?
- Ones that I know best
  - BASIC
  - Pascal
  - C (I initially hated it)
  - FORTRAN
  - Assembly Language



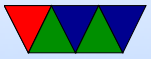
Other Programming Languages

- Low-level system languages: C, Rust, Zig, Go
- C related: C++, C#
- Interpreted languages: Java, Python, Javascript, Typescript
- Ancient ones: Fortran, Ada, COBOL, APL, Perl, Tcl, Pascal, BASIC
- And many, many more. . .



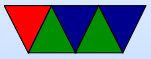
Can You Write your own Programming Language

- Yes!
- It's a bit beyond this class though, but anything a computer is doing someone had to originally program it to do so



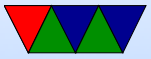
Hello World

```
/* Hello World Example */

#include <stdio.h>

int main(int argc, char **argv) {

    printf("Hello World!\n");

    return 5;
}
```



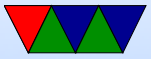
Livecoding Interlude

- It's hard to live-code on screen with people watching
- Some people do it as part of live Demoscene competitions
- Here are two example from demoparties:
  - Shader Showdown (in GLSL shader language which is vaguely C-like)
    `https://www.youtube.com/watch?v=JEn3kpysimY`
  - Byte-batte, a size-coding competition with a 256-byte limit, in Lua
    `https://www.youtube.com/watch?v=2JkD6JtzTW0`



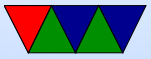
Hello World

- Typical C example program
- First hello world example saying that by Brian Kernighan in 1972 (in B language)
- There's already a lot going on here but you'll learn more than you ever wanted to as the course progresses



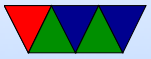
Compiling Code

- The computer you have can't run "high level" .c files like this directly
- The CPU in your computer only understands raw binary numbers
- You need a series of programs to convert this text file down to the "machine code" the actual hardware handles
- The actual hardware itself is fascinating, recommend "Computer Architecture" (ECE473) to learn how to build it yourself out of transistors



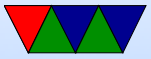
Compiler

- The Compiler takes high level text file, converts it to an executable that your operating system can pass to the processor to run.
- An executable is the raw machine code often with a header on it that tells the OS how to set it up before running (ELF on Linux)



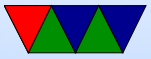
Compiler Info

- In the example we use “gcc”, the GNU C compiler which is a free compiler that is standard on Linux
- There are other C compilers. Another free one is “clang” that comes with LLVM
- In the old days there were many commercial C compilers you could buy, though they are less common these days. One common one was icc which was put out by Intel to work best on their chips.
- Compiler inner working are fascinating but beyond this class



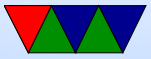
Compiler to Assembly Language

- Compiler converts text.c file to "assembly language" which varies by processor type (x86, ARM, etc)
- Assembly Language is one step up from the binary "Machine Code" the CPU uses. It still looks vaguely like English text, but each instruction maps directly to a low-level instruction
- Some people can write machine code directly, but most would prefer using higher-level assembly language.



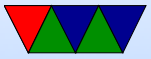
Assembler to Object Code

- An assembler converts the .s or .asm file to an "object file" (.o or .obj) which is mostly machine code binary
- Linux by default this is the GNU assembler (gas)
- Assemblers are much easier to write than compilers (it's mostly string parsing and bit shifting) though some architectures (looking at you x86) are hard



Linker to Executable

- There's one more step to take the machine code into a form that the Operating System can properly set it up and run it
- A linker converts the object file to the final executable (ld on Linux)
- This supplies a header (ELF on Linux) that tells the OS how to load the binary code and where to jump into it, among other things



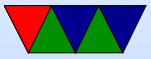
Compiling Hello World

- Can do the steps separately, or gcc can do it in one step
- Often you'll have a build setup that automates this
- On Linux often use the `make` tool and Makefiles
- To compile manually looks like this:

```
gcc -O2 -Wall -o hello_world hello_world.c
```

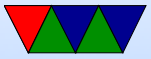
- If you want to generate assembly language to look at, you can do

```
gcc -O2 -Wall -S -o hello_world.s hello_world.c
```



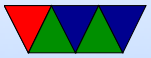
Source Code

- High level list of instructions you pass to a computer
- The higher-level you go often the more it looks like English with some symbols scattered about
- Usually is just a text file



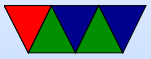
Text Files – What is a file?

- On most OSes it is a bunch of bytes grouped together and assigned a name
- What's a byte? An 8-bit value, 0..255 (0000.00000 to 1111.1111 in binary).
- Something called a filesystem tracks where the bytes are on disk and metadata (things like name, creation time, size)
- Generally all computers store chunks of data in files, though these days it might be hidden in the cloud

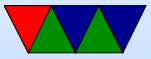


ASCII

- How do you get from 0s/1s to text?
- It's completely arbitrary
- Companies could map letters to numbers any way they want
- US Govt got fed up and in 1966 defined ASCII (American Standard Code for Information Interchange)
- man ascii
- Actually on 7-bits
- 0=48,1=49,..



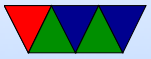
- A=65,B=66,..
- a=97,b=98,..
- Also “control” characters for things like line-feed, carriage return, new-page, etc



Hex Dump

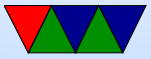
- Can use tools to dump the raw contents of your file

```
hexdump -C hello_world.txt
00000000  2f 2a 20 48 65 6c 6c 6f  20 57 6f 72 6c 64 20 45  |/* Hello World E|
00000010  78 61 6d 70 6c 65 20 2a  2f 0a 0a 23 69 6e 63 6c  |xample */..#incl|
00000020  75 64 65 20 3c 73 74 64  69 6f 2e 68 3e 0a 0a 69  |ude <stdio.h>..i|
00000030  6e 74 20 6d 61 69 6e 28  69 6e 74 20 61 72 67 63  |nt main(int argc|
00000040  2c 20 63 68 61 72 20 2a  2a 61 72 67 76 29 20 7b  |, char **argv) {|
00000050  0a 0a 09 70 72 69 6e 74  66 28 22 48 65 6c 6c 6f  |...printf("Hello|
00000060  20 57 6f 72 6c 64 21 5c  6e 22 29 3b 0a 0a 09 72  | World!\n");...r|
00000070  65 74 75 72 6e 20 35 3b  0a 7d 0a                 |eturn 5;.}.|
0000007b
```



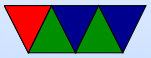
Control Characters

- Some "whitespace" characters with special ASCII values
  - space
  - tab (`'\t'`)– big debate about using these
  - carriage return `'\r'` / `'\n'` linefeed - indicate end of line
    date back to old typewriter days
    annoyingly Windows, Linux, and MacOS all used different combinations
- Could you make a computer language entirely out of whitespace?



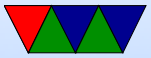
Text Editor

- Lots and lots of these over the years
- vi vs emacs
- nano/pico
- notepad?
- Something fancier in your browser?
- Possibly you use VisualStudio, that's fine, deep down it's just making .txt files too



What if you're not American?

- What if you want to type characters not in roman alphabet?
- on IBM-PC at least since ASCII only bottom 7-bits they tried to cram as many European chars as possible in top 128
- Not all fit. And what if you speak Russian (Cyrillic) or Chinese? Or want emojis?



Unicode/UTF8

- Why not use 16-bits? That gives you 65k characters
- Unicode. Windows and Java went with this
- Turns out 16-bits isn't enough, also not backwards compatible
- Plan9 people came up with UTF-8, a compromise
- If 0..127 just like ASCII
- If top bit set it means it is multi-byte
- Lets you encode arbitrary characters, downside is it makes parsing strings more complicated