ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 3

Vince Weaver

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

`vincent.weaver@maine.edu`

28 January 2026

Announcements

- Labs happening this week!
- Meant to start taking attendance this week but they keep adding more students. Maybe Friday.
- Homeworks, working on getting that set up

Brief Aside: Why C and not C++

- I hear C++ contains all of C plus all kinds of powerful extra stuff
- C is procedural, verbs, like printf(string)
  C++ is object-oriented, string.print()
- With C it's more obvious when you write code what the corresponding assembly will look like (c++ hides things)
- C++ adds lots of features to C that make things complex
  - Operator Overloading
  - Exceptions

  - STL
- For our purposes in ECE to have you experience a system language C is a much more straightfoward choice
- I also personally don't like C++ for various reasons but that's sort of tangential
- If you're a computer engineer you will encounter object-oriented in ECE277 (though in Python)

Lab1 Info

- I apologize this info is a bit late, was planning on giving it Monday but was thwarted by the snow
- Monday lab we are just going to delay to next Monday assuming it doesn't snow again
- Thanks to Tuesday lab for being the test-run
- If I duck out of lab it means I probably have a meeting, not that I am abandoning you

Lab1 What you Need

- Bring your laptop!
- If your laptop does not have a USB-A connector ideally you'd have some sort of adapter
- Currently there's no pre-lab
- The lab materials are posted off of the 177 website
- Lab submission is via Brightspace. The current 177 Brightspace page is not well populated because I don't have much experience with it

Lab1 Parts

- We're going to start handing out parts to you
- Don't lose them! You can keep them in the end. We don't really have any extra
- This week you get a breadboard/wires, Pi-pico, and USB cable
- You'll upload 6 screenshots to Brightspace

Lab1 Goal

- Install the tools on your laptop that you need for later labs
  - This includes a C compiler, and VS Code
  - Also PuTTY or `screen` to access serial port
- You will compile and upload sample Pi Pico code that blinks an LED and prints a value to serial
- You will also write a simple hello world program and run it locally on your laptop

Software Installation

- The assumption is you have a Windows Laptop and in theory those directions are the most tested
- We provide Mac and Linux directions too though it might be harder to get those going
- I personally am not a Microsoft fan so while I've tested the Mac and Linux I haven't tested Windows and am leaning on TAs for help with that

Why VS Code?

- Why are we using VS Code?
- Some people do like it. It's a “modern” IDE and widely used
- It's also the official supported way of doing things in the official Pi Pico documentation
- Previous years they've used it and I foolishly assumed that meant the directions were well-tested

Can I do things w/o VS Code?

- Maybe...
- I personally like doing everything at the text prompt and in fact have previously done some Pico programming that way
- The pi-pico SDK assumes you are using the “CMake” build system which is super-complex (alternatives aren’t necessarily better)
- If I have time I’ll see if I can come up with some VS-code less directions

Writing Hello World

```c
/* Hello World Example */

#include <stdio.h>

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

        printf("Hello World!\n"); // comment

        return 5;
}
```

C code – Comments

- It is *always* important to comment your code
- In C there are two ways to do this
  - Multi-line comments `/* ... */`
    anything in between ignored by compiler
  - End-of-line comments `// ...`
    anything after to end of line ignored
- Can also be useful to temporarily disable parts of your code when testing

Comments – What should they be like?

- Describe what the code is doing
- Say who wrote it, copyright, license
- Think, if I come back to this code in 6 months, will I still understand what it does
- If you share code with someone else, can they understand what it does?
- It is hard to over-comment code
  However don't just restate things

Comments Example

- Comments that just literally restate the code aren't always that useful

```
int i=1; /* set integer i equal to 1 */
\begin{lstlisting}
\item Better is if they explain the things that would be
\begin{lstlisting}
int j=2;  /* set the vertical index register equal to th
          /* value described in Section 4.2 of the manua
```

C coding style

- Are there any rules on how to write code? (not really)
- Does the compiler enforce things like Python does? (No)
- Most projects have a preferred style
  - New data blocks should be indented, either with tabs or spaces
  - Should the `{` be on a new line?
  - Should there be spaces after keywords?
  - Should you wrap text at 80 lines?
  - People will argue forever about these things

◦ VariablesCamelCase or lots_of_underscores or Hungarian notation (with type info)

International Obfuscated C Code Competition

- `https://www.ioccc.org/`
- In 80s people thought some C code was awful
- As a joke had a competition to see who could write the most confusing code
- Has been running off and on since
- A competition is currently open now (deadline in March)
- I have won twice (once in 2005, once in 2025)
- Please do not write your ECE177 code this way

Show Some Example Winners

- Showed 2012/endoh1 ("self documenting" fluid dynamics simulator)
- Showed 2005 "Most Beauteous Visuals" winner (3d-cube)
- Showed 2024 "Sur-Prize" winner (you'll have to go view it yourself)

C Variable Declarations

- Variables are values in memory used by your program.
- In C you need to declare them before you use them
- In old days had to initialize at the top of a block, these days not anymore
- You can specify a starting value, but you don't have to What value does it get if you don't?
- You can specify more than one on a line `char a=0,b,c=2;`

C Variables – Behind the Scene

- You can give names to your variables
- The hardware doesn't know about this, as far as it knows it just has addresses in memory storing bits
- The compiler is responsible for helping your code allocate memory for these, and mapping the name to the address in memory

C has various built-in data types made of one or more bytes

- To confuse things, these aren't necessarily the same size on different machines
- They also may come in `signed` and `unsigned` variants
- To the processor though it's just bytes moving around, it's up to you the programmer and the programming language to build up higher-order things

char

- `char a;`
- Often the size to hold a character for printing
  No longer true with unicode (windows has special wchar)
- Usually 8-bits
- Can be `signed` or `unsigned`.
  Annoyingly the default can vary based on what platform you are on.

short int

- `short s;` or `short int s;`
- signed / unsigned
- At least 16-bits. Usually 16-bits

int

- `int i;`
- signed / unsigned
- At least 16-bits. Usually 32-bits on modern machines.

long int

- `long l;` or `long int l;`
- signed / unsigned
- At least 32-bits. Usually big enough to hold a pointer?

long long int

- `long long l;` or `long long int l;`
- signed / unsigned
- At least 64-bits. Usually big enough to hold a pointer?

Rules of relative sizes

- sizeof(char) $\leq$ sizeof(short) $\leq$ sizeof(int) $\leq$ sizeof(long) $\leq$ sizeof(long long)

System Differences

- ILP32 (int, long, and pointers all 32-bits): 32-bit Linux, 32-bit macOS, 32-bit Windows
- LP64 (long, pointer 64-bits): 64-bit macOS, 64-bit Linux
- LLP64 (long is 32-bits, pointers 64-bit): 64-bit Windows

What if you want exact sizes?

- You can `#include <stdint.h>`
- Then you can use exact types:
  - `uint32_t x;` is a 32-bit unsigned int
  - `int8_t q;` is an 8-bit signed int

Floating Point

- By default a 32-bit int can be from 0...4GB or so, or -2GB to 2GB
- What if you want bigger or smaller? Fractional values?
- Floating point allows these, has a mantissa and exponent. How it's encoded is a bit beyond this class
- Can also encode special values like NaN (not a number) and +/- infinity

float

- `float f;`
- Generally 32-bit

double

- `double d;`
- Generally 64-bit

Aside

- x86 has long double? 80 bits?
- Modern day systems can have 16-bit, 8-bit, 4-bit, etc, for speed. Mostly find these on GPUs
- Supercomputers measure performance in FLOPS (floating-point operations per second)
  GPUs will report really high FLOP count by using the smaller sized floats

There can be a lot of complications to Using Floats

- In binary most fractional values end up not being exact but repeating decimals and so math tends to be a bit inexact
- Comparing with $=$ is dangerous, really need to compare within a range
- Rounding can cause problems, and some algorithms might not converge
- This is all a bit beyond this class

Sample Code

```c
/* test sizes of data types */
#include <stdio.h>

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

    printf("Size of char is: %d bytes\n",sizeof(char));
    printf("Size of short is: %d bytes\n",sizeof(short));
    printf("Size of int is: %d bytes\n",sizeof(int));
    printf("Size of long is: %d bytes\n",sizeof(long));
    printf("Size of long long is: %d bytes\n",
           sizeof(long long));

    return 0;
}
```

More Advanced Data Types

- We'll see later we can build more advanced data types based on the built-in ones described here
- We'll revisit this in the future