ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 11

Vince Weaver

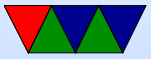
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

18 February 2026

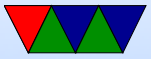
Announcements

- Don't forget HW#2 due tonight on codelab
- I will assign HW#3 when I get the chance.
  Note it's not necessarily one of the assignments already there so be careful working ahead



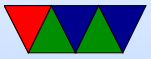
Lab3 Notes

- Be sure to bring a laptop and your assembled breadboard to each lab
- I know it is difficult to drag it around, it might be possible to improvise a case for it
- Try not to bend the keypad ribbon cable to hard. Tempting to crease it, but that can break the wires inside.
- Can you damage either your parts of laptop with improper wiring?



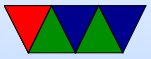
Breadboard Challenge

- This year's breadboards different than last year, meaning lab procedures had to be updated at last minute
  - Split power rails (need to bridge with jumpers)
  - Row/Column grid completely broken.
  - Columns indexed from -2?
  - Reversed row markings on right/left
  - Reversed column markings on top/bottom
  - Extra column of connectors
- Also colors of jumper wires different



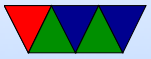
Note it's the connection that's important, not the color, just pick one roughly the right size

- I've updated the lab documents
- Best bet is to do things based on the images rather than the row/column descriptions



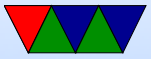
Breadboard Trivia

- Showed comparison image of old/new breadboards
- Showed some example rats-nest breadboards from older projects of mine
  - BTTF Time Circuits Project
  - Chiptune Music Player Project



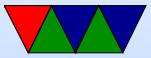
Detecting Even and Odd Numbers

- I said using a big if/then or case/switch probably not optimal
- Might not be too bad for char (256 lines)
- Would be tedious to write this all by hand, but could you write a program to generate C code that you could then compile (yes!)
- Would be interesting to benchmark speed



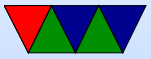
Could You Brute Force the 32-bit (4+ billion) case?

- could you handle that many cases?
- Could you write the code (can you write a program to write a program. . .)?
- Would the C compiler work on something that big?
- Would your OS run the resulting executable?
- Would you have enough RAM?
- Morbidly curious how it would work



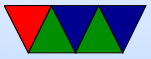
More Practical Ways – Mod

- `odd=value%2;`
- Slow on most machines as this is a divide instruction (and possibly an additional multiply and subtract) and division is generally slow



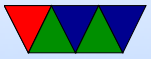
Related Way – And

- In binary, if low digit 1 is odd, 0 if even
- Can mask off with bitwise and
- `odd=value&1;`
- NOTE: If doing a MOD of a power of 2 (say X), it turns out you can use an AND of X-1 for the same result your compiler is usually smart enough to do that for you



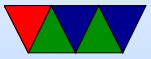
Other Ways – Shifts

- If you shift right and check which bit shifted off can tell
- This is easy in assembly where you shift to Carry bit
- A bit harder in C, you might have to shift right, shift left, then compare against original value

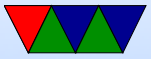


Other Ways – Divide by 2

- This is just a variant on the shift
  which makes sense as a shift is same as dividing by 2

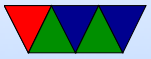


New Topic — Arrays



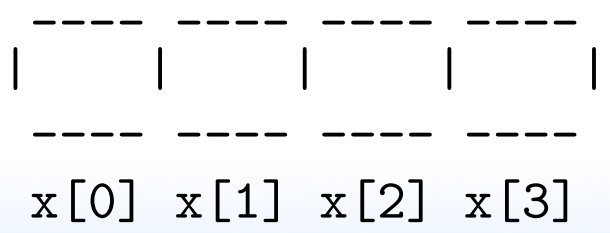
Arrays

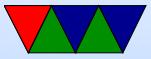
- Group of same data type stored consecutively in memory
- Sort of like a vector in math
- Note: unlike some other languages, C has no built-in vector type



Array Declaration notes

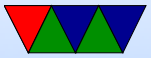
- `int x[3];  // declare array of 3 ints`
- type, then variable name, then number of elements in square brackets
- The example above has 3 elements, from 0..2
- C always indexes starting with 0 (some languages start with 1) : **CAREFUL! This can be a source of errors**





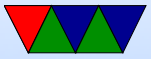
Accessing Array Contents

- `int x[3];`
- `x[0]=5;` Set 0th element of array to 5
- `printf("%d\n",x[2]);` print 3rd element of array



Array Bounds

- `int y[15];`
- Can you access `y[0]`?
- What about `y[500]`?
  C will let you do this, possibly without warnings.
  A bad idea. What happens if you do this?
  Best case bad value, worst case crash program or worse
- What about `y[15]`?
  Careful, that's actually out of bounds (0..14 are valid), easy mistake to make



Array Initialization

- Can initialize at declaration

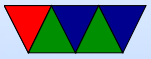
```
int x[3]={ 0, 1, 2};
```

- Can initialize manually

```
int i,x[3];
for(i=0;i<3;i++) x[i]=i;
```

- What happens if you don't initialize?

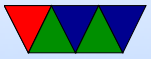
  Depends on various things, but you might get arbitrary garbage in it



Multi-dimensional Arrays

- 1D array, like a line (vector), one row of columns
- 2D array, like a rectangle (matrix), multiple rows of columns
- 3D array, like a cube
- you can have as many (?) dimensions as you want, though after 3D it can be harder to think about

```
int x[4];
int y[4][3];
int z[3][5][9];
int a=z[1][2][3];
```

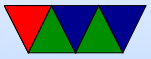


Multi-dimensional Init

```
char y[2][3]={ {1,2,3},{4,5,6}};
```

Note, you can have trailing commas in C, this sometimes makes auto-generating code a bit easier:

```
char y[2][3]={ {1,2,3,},{4,5,6,}};
```

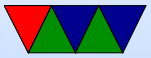


Multi-dimensional Layout

```
int a[4][3];
```

Note if you think of it as 2-dimensional x (horizontal) y (vertical) it's actually flipped from what you expect, it's `a[y][x]`

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]



Array Aside

- Arrays in C are row-major. Meaning in 2-dimensional array, row elements are next to each other in RAM
- Some other languages (notably FORTRAN) are column-major
- This can actually drastically impact performance on a system due to CPU caches (although things like prefetchers may hide some of this)