ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 16

Vince Weaver

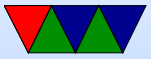
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

4 March 2026

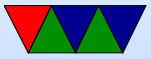
Announcements

- HW#4 due tonight, HW#5 will be assigned
- People seem to be doing OK on HW#4
- I'm going to try to put some relevant midterm questions on HW#5 if you want some extra practice
- ECE department is sponsoring a midterm review tonight, 5-7pm Barrows 130, there will be pizza



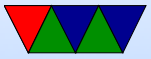
A few last things on structs

- We discussed structs last time. Note: they will not me on this midterm
- Question: are structs like C++ or Java classes?
  - structs are much simpler. Just groups of variables. No methods or private/public
  - You *can* if you want to include functions in a struct To do that involves declaring pointers to functions and assigning the address to functions
  - That is over-the top complex (and the syntax is a

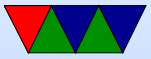


nightmare)

Some advanced projects like the Linux kernel do use it.

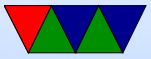


A few extra things on Lab#5

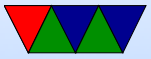


Lab#5 – Constants with Single Bits Set

- We request you use `(1<<13)` instead of `0b0010.0000.0000.00` or `0x2000`
  (Note, don't put . in large binary numbers, I'm doing that to make it easier to see)
- This is why binary was only a late addition to C because once you are above 8 bits it becomes unreadable
- Also note: GPIOs (and binary numbers in general) usually numbered starting with 0 so GPIO13 matches `1<<13`

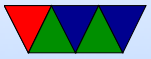


GPIO0 is `1<<0` which is 1 (bit 0)
GPIO1 is `1<<1` which is 2 (bit 1)



Lab#5 – Setting/Clearing Bits

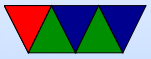
- How can you set bit without touching others?
- Why not touch other bits? In this case might flip GPIO doing something else important
- Probably won't destroy your Pico, but it could make display/ keypad not work until you reset it
- To change things, need to do a read/modify/write



Lab#5 – More Setting/Clearing Bits

- This example assume value you want to change in x and there's a temporary variable y
- We want to change bit 26, so set y to an int with only bit 26 set

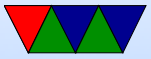
  `y=(1<<26);`
- To set this bit in integer x use bitwise OR
  Set bit, `x=(x|y);`
- To clear this bit in integer x, use bitwise AND of the inverted y (creates what's called a MASK where a bit of



1 means keep old value, a bit of 0 means clear it)

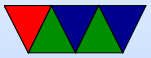
```
x=x&(~y);
```

- You might then want to shift left, `y=y<<1;`



Lab#5 – Setting/Clearing Advanced

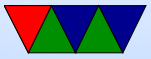
- There are fancier/cleverer ways to do this
- While you are learning we prefer you do things the straightforward way
- Sometimes might seem inefficient, like having an extra temporary variable
- Optimizing C compilers are smart enough to optimize for you
- Other ways to do things:
  - Could use XOR to toggle bit and leave others alone



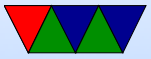
- Could have loop be i from 13 to 22 and use `(1<<i)` for the bit to set
- Could even have the loop be something like

```
for(i=(1<<13);i<=(1<<22);i<<1) {
```

- Again, in this class, the simpler way is usually preferred, if only for debugging and/or grading purposes

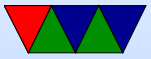


Review of things on Midterm



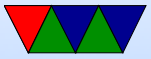
Number Systems

- binary / octal / hex numbers (no octal on midterm)
- Convert 0x32 to binary
- Convert -1 to two's complement binary
- decimal to binary: good to know
  If I do ask it won't be anything complex.
- binary to hex
- NOTE: in past years they spent a lot of time on binary arithmetic including addition/subtraction. I haven't really covered that so it won't be on the test

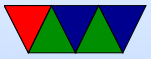


Variable Types

- Variable types, char, short, int, long, float, double
- additional specifiers like signed / unsigned
- Define a variable that can fit 3.14159
  double. float also ok (less precise)
- Define a variable that can hold 255
  unsigned char or larger. Often in C you use "int" by default unless there's a reason to use something smaller
- Define a variable that can hold -1
  needs to be signed

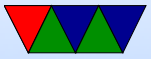


- Define a variable that can hold 'Q'
  usually you'd use char for this



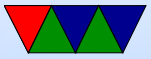
printf()

- `printf()`
- I'm probably not going to ask a specific question, but be sure to know:
  - Takes a string argument
  - This string can have replacement placeholders like `"%d"` which say to substitute in a value from the parameter list
  - For each substitution there needs to be a parameter
  - Common ones are i or d for integer, s for string, c for



ASCII char, f or lf for floating point

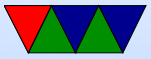
- There are more advanced things you can do but I won't be asking about that



Loops

- Loops for/while/do-while
- Write a for() loop that prints ECE177 9 times.
- Make an infinite loop
- For loops most common. While loops mostly equivalent. Do while loops will always run loop body once, for/while might run 0 times if the condition starts out false
- examples

```
int i; for(i=0;i<100;i++) printf("Hi\n");
int x=0; while(x<100) { printf("Hi\n"); x++}
int y=0; do {printf("Hi\n"); x++;} while(x<100);
```

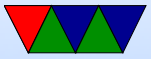


Flow Control – if/else

- Allow taking multiple paths through program based on conditions
- Example: have variables x and y, print Yes if x is equal to four and y is less than 10, otherwise print No

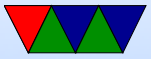
```
int x=4,y=3;
if ((x==4) && (y<10)) printf("Yes\n");
else printf("No\n");
```

- You can nest these arbitrarily deep



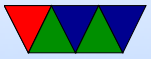
Flow Control – switch/case

- Can replace a long string of if/elses with switch case, where you "switch" on an integer value and have different cases as appropriate
- there's a `default` case at the end to catch any that didn't match
- After each case you need a `break`, otherwise you can fall-through and run the code in the following case. Sometimes you might want to do that intentionally but usually you don't.



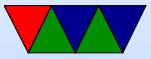
```
int x;
switch(x) {
     case 1: printf("1"); break;
     case 2: printf("2"); break;
     default: printf("Not 1 or 2\n"); break;
}
\end{list1}


\foilhead{\color{umblue} Arrays}
\begin{list1}
     \item Declaring arrays
          \begin{lstlisting}
          int a[5];
          a[5]=1;
```

- How to initialize array?



You can use a loop, indexing with a variable

- How to print out all values in array?



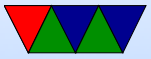
Strings

- Array of char

```
char str[10];
```

- C has NUL terminated strings
- To get size use `strlen(str);`
- You cannot compare with equals

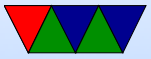
```
char str1[]="Hello",str2[]="World";

if (str1==str2) printf("Same\n"); // won't work

if (!strcmp(str1,str2)) printf("Same\n"); // proper way
```



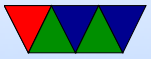
Functions

- Sub-routines you can call
- Need to be declared before you can call them
- Optionally can return one value
- Can take an arbitrary number of parameters, passed in by value (usually)

```
int sum(int a, int b) {
    int total;
    total=a+b;
    return total;
}

void some_other_function(void) {
```



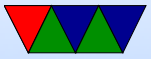
```
    int result;
    result=sum(3,5);
    printf("%d\n",result);
}
```



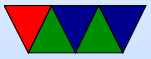
Bitwise vs Logic Operators

- This can be a bit confusing
- Bitwise operate on every bit in an integer
- Logic operate in a boolean (true/false) fashion where 0 means false and non-zero means true
- If you use the wrong one compiler might not complain and it might even sort of work but only due to luck

	bitwise	logic
and	&	&&
or	|	||
not	~	!
xor	^	

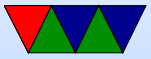


shift << >>



Bitwise Operators

- You're seeing this in Lab#5
- OR `|` is generally used to set bits
- AND `&` is generally used to mask, or clear bits (often in conjunction with not)
- XOR `^` is generally used to toggle bits
- NOT `~` will flip all bits.
- SHIFT `<<` or `>>` will shift bits left or right by a count value



Logical Operators

- Boolean Logic, same as bitwise but on a single bit (true or false)
- OR `||` is used to see if either case is true

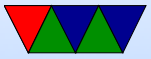
```
if ((x==4) || (y==5))
```

- AND `&&` is only true if both cases are true

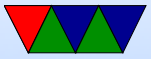
```
if ((x==4) && (y==5))
```

- NOT `!` will flip a result from true to false, or false to true

```
if (!(x>4))
```



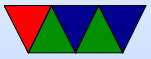
note you can do transforms on this, beyond this class



Boolean in C

- No dedicated Boolean type, just 0 or non-zero
- Any non-zero value is considered true

```
int cookies_left=5;
if (cookies_left!=0) printf("There are cookies left\n");
if (cookies_left) printf("There are cookies left\n");
```



Things we've covered not on this test

- In theory anything from Lab, Lecture, or Homeworks can be on the test
- Some things that won't be (or if they are, as extra credit):
  - C history
  - Pointers
  - Structs
  - Sierpinski triangles