ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 7

Vince Weaver

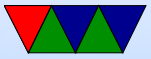
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

6 February 2026

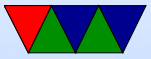
Announcements

- REMINDER: No food or drink in Labs!
- Accessibility office looking for note-takers, they said they e-mailed and no one volunteered so they asked me to mention it
- Homeworks have been set up in codelab!
- Learned about printf last time, technically Linux printf is Turing Complete



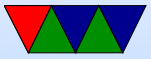
Brief Lab #1 Update

- If still having issues with Pico, now is the time to get them solved
- If it's the Ninja.exe error, the fix seems to be to *completely* blast away VS Code and the pisdk (possibly by force-erasing the .pi-sdk and .vs-code/extensions/pico* directories) and reinstall from scratch. Most people having this issue possibly already had VS Code set up previously and somehow the directions we followed messed it up somehow.

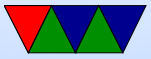


Brief Lab #2 Update

- I fixed the negative number problems. I've gone in and manually regraded all the people from Tues/Wed labs.
- Some people still didn't get 100% but it's mostly forgetting 0b or not having enough leading 0s on a 16-bit number
- Also, if this seems like I have Brightspace set up in an annoying way it's probably incompetence not malice. There are like 10 ways you can pick to handle a completed assignment with errors and I picked the one that seemed

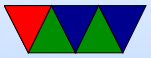


like the least hassle but maybe it wasn't the best choice



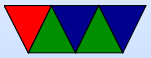
Setting up CodeLab

- Go to `https://codelab.org`
- Register using the directions and access code I sent in the e-mail
- It is a few annoying steps, to get the account made (e-mail verification and such)
- They ask for your birthday but claim they don't keep it
- Also it should *not* ever ask you for money, if it does let me know

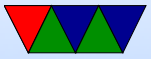


How it Works

- I'll set up a series of questions as Homework #1 with a due date (currently February 10th)
- Log in and on the side there will be a list of exercises
- Click on one, and it should bring up a question, like "Enter a literal of the number 2"
- if the answer is 2 then put it in the box and click submit
- If you're right you're good, move on to the next one
- If you're wrong it will give you feedback and you can try again

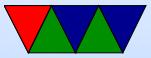


- You can try as many times as you want
- There isn't necessarily one right answer, for example 2 and 0x2 and 02 might all work

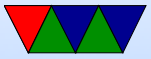


Constants

- What if you have a variable, but you don't want it to change?
- You can declare it as a constant
- This can keep you from accidentally changing it
- The compiler might also be able to put it in read-only memory
- Can use with any type
- Use the keyword `const`
  - can be before like `int const x;`

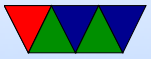


- can be after like `const int x;`
- can declare multiple `const int x,y,z`



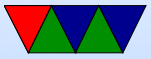
const Example

```
int i=5;
const int k=4;
int const o=5;  // also allowed
const float pi=3.14159265389793238 4;

printf("%d %d\n",i,k);

i=3;
k=2;     // will cause compiler error
printf("%d %d\n",i,k);
```

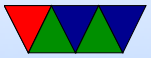


Statements

- Hard to describe what's going on without sound like a spec or compiler document
- A statement is a bit of text, separated by semicolons
- `x=y+1;` is an example
- `;` (an empty statement) is allowed (though be careful, can cause issues)
- `x+y;` is a statement, but since the result isn't used/assigned to something the compiler will warn (but it will compile)

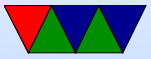


- `4;` is also a statement for similar reasons



Blocks / Compound Statements

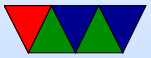
- A bunch of statements grouped together
- Surrounded by { } curly brackets
- All statements in block run one after another until done



Loops

- What if want to do things multiple times?
- Cut and paste?

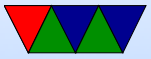
```
printf("Hello #1\n");
printf("Hello #2\n");
printf("Hello #3\n");
printf("Hello #4\n");
printf("Hello #5\n");
printf("Hello #6\n");
printf("Hello #7\n");
printf("Hello #8\n");
```



For Loop

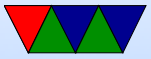
```
int i;
for(i=1;i<9;i=i+1) {
    printf("Hello #%d\n",i);
}
```

- Three things in the loop statement
  - Take variable and initialize it at start
  - (code block is run here)
  - Condition to check after each iteration to see if loop is done
  - Action to take before re-running loop



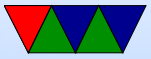
Aside on conditionals you can check

- These are boolean, resolve to true or false
- Greater than `>`
- Less than `<`
- Greater than or equal `>=`
- Less than or equal `<=`
- Equal `==`
- Not-equal `!=`
- We'll find out you can be overly clever here too



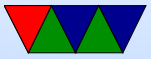
Other things you might see

- Why are iterators often i?
- Instead of `i=i+1` often you'll see `i++` instead. It means the same thing
- You might see `for(int i=0; i<10;i++)` where you declare the iterator inside the for statement
  This is a relatively recent C++ type thing to do but it is valid in modern C
- The various parts of the for statement can be sort of arbitrarily complex (We may see this later). For now it's



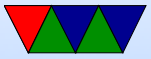
best to keep it simple

- We say we should have a { } code block, but if you only do one thing you don't need to have a block (but it's strongly recommended)



For Loops: Messing with the Iterator

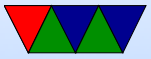
- What happens if you change the iterator in your program? If you're using i, what if you set it really high?
- This is allowed, but this is usually frowned upon because it makes it harder to keep track of what's going on in your program
- What value does the iterator have at the end of the loop?



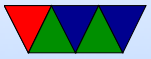
While Loop

- Set up an iterator in advance
- Checks the exit condition
- If not true will run code in a block
- It's up to you to update the iterator before the end of the block
- Loops back to the part where it checks the exit condition
- You can see while and for loops are similar and in fact behind the scenes the compiler treats them the same

```
int x;
x=0;
```



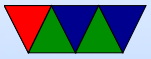
```
while (x < 10) {
     // do something
     x=x+1;
}
```



Do While Loop

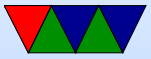
- The main difference is that it will always run at least one loop iteration
- With a while loop if the condition is not true at the start then it won't run at all.

```
int x=0;
do {
     // something
     x=x+1;
} while (x<10);
```



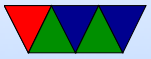
Exiting Loops Early

- `break;` will exit out of the loop you're currently in
- NOTE: you can have loops inside of loops, `break` only gets you out of the inner-most one you're in
- `continue;` will skip the rest of this current loop iteration and go to the beginning of the next iteration

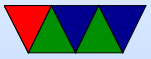


Infinite loops

- `while(1) {`
- `for(;;) {`
- Why would you do this?
  Might you have a loop that you don't know how many times it will be in advance? Maybe reading input from keyboard?
- How can you exit? `break`
- If your computer is stuck in one at the console, on Linux at least Control-C will force the program to exit

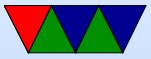


(otherwise, force-kill?)



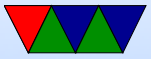
Functions

- How you can split up work in C
- Named block of code
- Can pass in paramaters
- Can return one value



Function Example

```
int sum(int a, int b) {
    // note a and b passed by value
    // you can change the value and it won't affect
    //  anything in the calling function

    int result;

    result=a+b;

    return result;

}

int main(int argc, char **argv) {
    int c;
```



```
    c=sum(a,b);
    printf("Result is %d\n",c);

    printf("Can also do: %d\n",sum(1*2,3+4));

    return 0;
}
```