ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 18

Vince Weaver

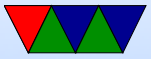
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

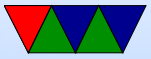
11 March 2026

Announcements

- No, the midterm isn't graded yet.
  It's unlikely it will be graded before spring break.
- Still working on HW#5
- Reminder, No Lab#6 this week!
  Makeup lab. Please show to a lab session (Tues/Wed/Thurs, 2-5pm) if you have outstanding labs that need to get checked off.



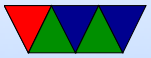
Some Last Preprocessor Stuff



Over-the-top define replace

- German for “if” is “wenn” (oder is “or”)

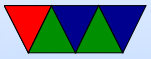
```
#define wenn if
#define zurueck return
#define schreiben printf
#define oder ||
#define primaer main

int primaer() {
        int x=4,y=5;
        wenn ((x==3) oder (y==5)) {
                schreiben("Gut!\n");
        }
        zurueck 0;
}
```



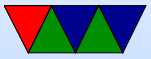
assertions

- You can `#include <assert.h>` to get access to the `assert()` macro
- Put this in your code to sanity check values/invariants
- If they fail, it will exit your program with an error and say what file and line number
- Often people do this when debugging code, then disable them when they "release" the production code

```
int month;

some_code();
```



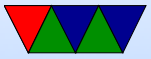
```
assert(month<=12);   // month should never be greater than
                     // if it happens something is very wro
                     // and we should stop the program
```



Disabling Code

- You can use `/* */` to disable code
  But what if comments inside of comments?
- An alternate way to disable code temporarily is using `#if 0` / `#if 1` with the preprocessor

```
#if 0
  printf("don't print this now\n");
#endif
```

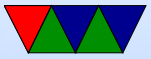


Type Awareness of Macros

- Macros are sort of type-agnostic

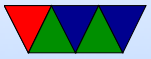
```
#define SQR(a) ((a)*(a))
```

  works for both ints and floats



The Remaining C Operators

- It's almost time to discuss the full "order of operations" precedence in C. It's quite complex
- Before we do that there are a few last operators we haven't talked about yet, some of them obscure



How are Commas Used in C

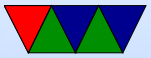
- So far you've mostly seen them to separate arguments to functions

```
printf("Hello %d %d\n",x,y);
```

- You use them for something similar when defining preprocessor macros

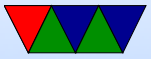
```
#define SUM(a,b) ((a)+(b))
```

- Used to chain a number of expressions together. (this is a bit obscure)



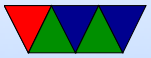
The Comma Operator

- Lets you chain a bunch of expressions into one big expression
- This is most useful if the C standard expects a single expression in some place but you want to have it do multiple
- `expression1,expression2;`
- Evaluated left to right
- expression1 is carried out, then expression2. The overall construct gets the value of expression2, so if



it's `x=(a=4,y=5);` then a gets set to 4, y gets set to 5, then x gets set to 5

- *NOTE* comma operator has the lowest priority of any operator, so `x=4,y=5` is equivalent to (x=4),(y=5) not x=(4,y=5)



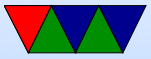
The Comma Operator – Most Common Use

- You'll most often see this when jamming lots of stuff into a for loop
- Instead of

```
sum=0;
for(i=0;i<10;i++) {x++; sum++;}
```

- You can do

```
for(sum=0,i=0;i<10;i++,x++) sum++;
```

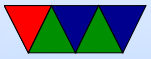


The Comma Operator – Dangers

- Note that C does not let you use comma as a thousands separator, even though it would make things more legible
- This is more trouble if you're European and use comma as a decimal point

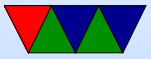
```
int mile;
mile = 5,280;
```

- What does this do?
  - It will store 5 in mile
  - The overall expression evaluates to 280 (which is true)



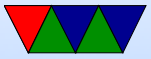
Splitting Up Constants

- Possibly C23 adds support for using ' for this
- `0xAB'CD'EF'12`
- `0b0000'1010'0101`
- Unfortunately you have to wait until all systems get C23 support



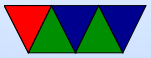
Ternary (?) Operator

- Special conditional operator in C
- Only one that takes 3 arguments
- Lets you do some complex, powerful things in a small amount of code
- Mostly for advanced use, but you will see it in action



Ternary (?) Operator

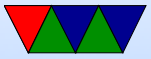
- `expression1?expression2:expression3;`
- Test expression1, with a true/false answer
- If true, then the whole thing gets the value of running expression2
- If false, then the whole thing gets the value of running expression3



Ternary Example

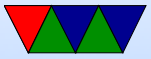
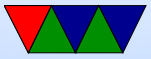
- Absolute value

```
int x;
x=(x>0)?x:-x;

Equivalent

if (x>0) x=x;
else x=-x;
```



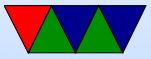
Another Example

- ```
  int x;
  x=(a>0)?b:4;

  Equivalent

  if (a>0) x=b;
  else x=4;
  ```


  ```

Ternary Example – Strings

- ```
  int cookies;
  printf("You found %d cookie%s\n",cookies,
                                   cookies>1?"s":"");
  ```


  ```

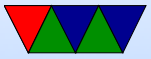
Ternary: Advanced

- Can you have nested ternary expressions?
- Yes, but that's heading to IOCCC territory



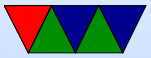
sizeof

- Can be used to get the size of a variable or type
- Returns number of bytes



sizeof type

- Can do things like `x=sizeof(int);`



sizeof variable

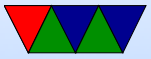
- Can do things like

```
int x;
printf("Size of x: %d\n",sizeof(x));
```

- Note with variables the parenthesis are optional

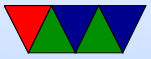
```
y=sizeof x;
```

- This is not true for types



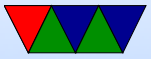
sizeof arrays

- `double a[10]; sizeof(a);`
- `sizeof(double[10]);`



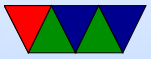
sizeof structs

- Can use to find size of structs
- NOTE: sizeof structs might be larger than the sum of all the elements due to PADDING
- You can create a packed struct to avoid this (how)
- Normally you wouldn't care, but if you're trying to map struct to hardware registers, or a cross-platform binary format it does come up
- See next lecture for more info on padding in structs



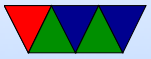
sizeof unions

- This might be complex, we'll talk about it if/when we discuss unions



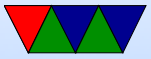
sizeof – Cautions

- Cannot use to find size of string (use `strlen()` for that)
- If you do use on string, on array is total size of array, if a pointer will probably get the size of a pointer



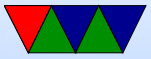
size_t

- How big can the result of `sizeof` be?
- On 32-bit system probably 4GB, but on 64-bit much larger
- In that case an `int` might not be big enough to hold it
- C has special type `size_t` that is always guaranteed to be big enough to hold the size of the biggest possible variable



Casting

- C is not a strongly typed language
- It is possible to assign variables from different types to each other
- This conversion from one type to another is called casting

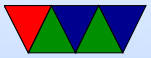


Implicit Casting

- As we saw before, you can do things like

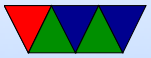
```
double f=4.5;
int x;
x=f;
printf("%d\n",x); // should get 4
```

- C will automatically truncate / convert among the various numerical types

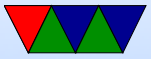


Downsides of Implicit Casting

- You can truncate or lose precision



Explicit Casting

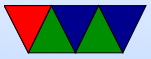


Advanced Explicit Casting

- While C is not strongly typed, it will in some cases warn/error if you try to force something from one type into another

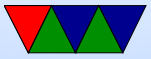
```
int *pointer;
int x;
pointer=x;
//gets you  error: assignment to ‘int *’ from ‘int’
// makes pointer from integer without a cast [-Wint-conv
```

- Could you force this anyway? `pointer=(int *)x;`
- Yes, BUT ONLY DO THIS IF YOU HAVE A GOOD REASON



Good Reasons to Cast

- Force numerical values to a different type before doing math on them
- After allocating memory, setting to the proper type



Bad Reasons to Cast

- To make compiler warnings/errors go away