ECE 177 – Programming I: From C Foundations to Hardware Interaction
Lecture 25

Vince Weaver

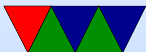
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

3 April 2026

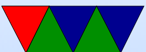
Announcements

- HW#6 was officially posted, due Monday
- Watch for HW#7
- Reminder midterm the 10th
- Lab#8 Next Week



Midterm2 Topics

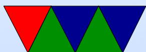
- structs
- scanf()
- string/string comparisons
- pre-processor
- pointers
- commas, ternary, sizeof, casting, goto
- operator precedence
- material from labs



HW6 Update – Pointer Arguments to Functions

- Normally in C when you pass parameters to a function it "passes by value": a copy is made of the values and the function gets the copy. Any changes to this copy are local and do not affect values in the calling function
- Aside: the exception to this is arrays, if you pass an array it is "passed by reference" (essentially a pointer is passed)

```
int add(int x, int y) {
  x=x+y;    // x is modified, has no affect
```

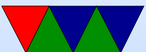


```
            // on values in main()
  return x;
}

int main() {
   int a=3,b=5;
   add(a,b);     // copies of a and b passed to add
}
```

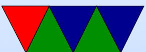
- If you want to pass in values that can be changed, use pointers

```
int add(int *x, int *y) {
  *x=*x+*y;   // the value pointed to by x modified
              // this will change the value in main()
  return *x;
}
```



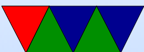
```c
int main() {
    int a=3,b=5;
    add(&a,&b);        // pointers to a and b passed in
                       // the add() function can change
                       // the values of a and b if it uses
                       // these pointers
                       // a will be 8 after running this
}
```

- You can mix and match regular vs pointer arguments in functions, depending if you need to have the function modify values from the calling function



HW6 Update – Pointer Swap

- Question 10243 is essentially asking you to swap two pointers
- It is missing "was pointing to" from the end
- How do you swap two variables in C?
  - You can't just `x=y; y=x;`
  - Usually you'd just have an extra temporary variable
  - ```c
    int x,y,temp;
    x=4; y=5;
    /* swap x and y */
    temp=x;
    x=y;
    ```

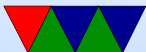

    ```

```
y=temp;
```

- Is it possible to swap without using a temporary variable?
- This is a common interview question. It is using fancy use of xor but it almost never makes sense to do this.

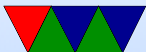
```
x=y^x;
y=x^y;
x=y^x;
```

- Many processors have an instruction that will do this in one step (`xchg` on x86) but this isn't really exposed to C



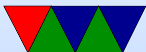
HW6 Update – More typos

- 10171 – says `,"ient,` instead of what I suppose should be `,"ient`
- In theory in HTML you escape things with ampersand and so it's plausible `"` would become " but weird
- I should figure out who to report typos like this to



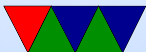
HW6 Update – Pointer Arithmetic

- We'll discuss this later in the lecture



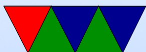
free()

- When you are done with your memory you need to free it
- What happens if you forget to free it?
  Can be a memory leak
- What happens if you accidentally free it twice?
- Note that if you have an OS like Linux, if you exit your program all memory is freed so a memory leak doesn't happen there. Because of that some people are lazy about freeing memory.



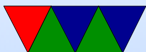
Garbage Collection

- Other languages support dynamic memory but they don't make you worry about freeing and such. How does that work?
- They can have "garbage collection"
- When you allocate an object you don't get a pointer. The object has a reference count
- When you are done using it this goes to 0
- Occasionally your program will stop and the garbage collector will run which will automatically free anything



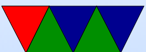
unused

- This can be slow or unexpectedly take time, but maybe worth it
- Also can compact memory to avoid fragmentation
- You can't do that on C because you never know who has a pointer to an object so it's stuck where it is forever



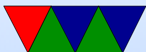
realloc()

- `void *realloc(void *pointer, size_t size);`
- What if you want to change the size of your memory allocation? It got bigger or smaller?
- If pointer is NULL, it's the same as malloc()
- If it's smaller it will be left in place but
- It it's bigger it might have to allocate a new chunk of RAM and then copy the old values over (it does not zero the new empty space)
- If you realloc() to 0 size if frees it



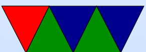
Algorithms Using Dynamic Memory

- You might be used to other languages that are constantly allocating and freeing memory
- This is less common in C just because of how hard it is to get right
- Also in the old days this wasn't very high performance



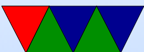
Debugging Dynamic Memory

- Very easy to have memory issues
- On Linux there's a tool called "Valgrind" that will help you debug these issues
- It's a dynamic binary instrumentation tool, re-compiles your code on the fly to add instrumentation to check for errors, but makes it run much slower
- My Valgrind story



Can you use malloc() on the pico?

- Yes
- But beware, only so much RAM so easier to run out
- For example, if you wanted to draw graphics offscreen for Lab, you might allocate 16-bits*320*200 for this which is 153k but the Pico only has 264k so tight fit
- Though it helps that executables on the Pi XIP (execute in place) from the 2MB of flash



Pointer Math

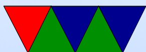
- If you have an pointer to an array or an allocated region, you can treat it as an array to access the data, i.e.

```
char *array=malloc(1024);
array[5]=3;
```

- You can also use "pointer math" instead. So to get the 5th element, just add 5 to the pointer

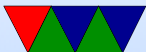
```
char *array=malloc(1024);
*(array+5)=3;
```

- You can add or subtract integers to a pointer and it's like indexing



- The C compiler knows the size of the elements of the array/pointer from when you declared it. So

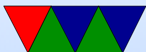
```
int *array=malloc(10*sizeof(int));
*(array+5)=3;
// this indexes 5 integers in (20 bytes) not 5 bytes
```



Aside, Array/Pointer equivalence

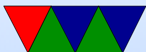
- Arrays are sort of just pointers

```
char *ch=malloc(10);
ch[5]=1;
*(ch+5)=1; // equivalent
5[ch]=1;   // bizarrely this works too
           // it converts to *(5+ch)=1;
```



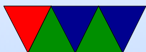
Pointer math in the HW

- Why does it go on and on saying things like there are 100 elements and one pointer is the first half and such?
- In theory pointer math is only valid in certain specific circumstances
- If you go out of bounds with things it can become undefined behavior



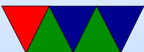
More Pointer Math

- Generally you can add an integer to a pointer, because it's the same as indexing an array
- Note it knows the type of the pointer, so if you have an integer pointer and add one to it, it actually adds 4 bytes to it (one integer worth)
- You can increment too
- You can also subtract



Pointer Math Limitations

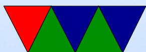
- You really shouldn't add/subtract two different pointers.
- Why would you want to? Maybe try to figure out how far apart in RAM they are? But really there's not a good reason to
- According to C standard you can't add/subtract VOID pointers but GCC lets you (it assumes a size of 1 byte)



Address of first element of array

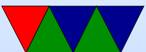
- The value of an array is the same as a pointer to the first element

```
int a[100];
int *b;
b=a; is same as b=&a[0];
```



Can you point beyond end of array?

- Yes, you are allowed to point one past the end
- This will happen if you use a pointer in a for loop, for example
- You area allowed to point to this, but it's not allowed to dereference it



Pointer Comparison

- You can check if equal or not-equal
- You can also check greater than or less than, though again ideally it would be with pointers inside the same object, and also again why would you want to do this