ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 24

Vince Weaver

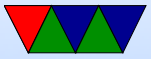
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

1 April 2026

Announcements

- Sorry behind on things, will try to catch up
- HW#6 will be posted, deadline extended until Monday
- Reminder midterm the 10th, more soon

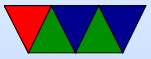


HW6 Update – Preprocessor

- Macros: put lots of extra parenthesis.

  ```
  #define PRODUCT(x,y) ((x)*(y))
  ```

  This is because x or y can be statements that can expand and cause trouble, say you run `PRODUCT(a+3,n+2)` and you get `a+3*n+2`
- How do you find out what include files to include? manpage? google?
- How do you have two things happen in one macro? In theory can just have one after the other on same line.



To be extra safe you can add in curly brackets I think

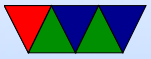
```
#define TWO_THINGS thing1(); thing2();
```

- Remember quotes vs angle brackets difference in includes
- Using ternary in macro, something like

```
#define MIN(a,b) (a<b)?a:b;
```

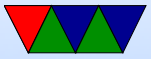
But you might need more parenthesis for order of operations reasons.

- Someone had good question, can you put if/else in a macro? Maybe but if things getting that complex you might as well use a function (or inline function)



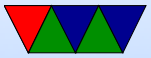
HW6 Update – Strained Pointer Analogy

- Analogy: normally declare a variable by name, compiler remember where it lives `bobs_house=5;` ask compiler to stick 5 in bobs_house
- You can also get the address of bobs_house, 123 main st. This is a pointer, and you can use that to find bobs_house yourself to put something there by following the address/pointer.

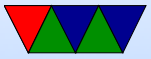


HW6 Update – Pointer Syntax

- Declare an int variable `int x;`
- Declare a pointer to int `int *y;`
- Taking address of a variable `y=&x;`
- De-referencing `x=5;` same as `*y=5;`
- We'll talk about the other stuff needed for homework next class

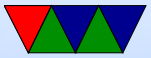


Lab7 – More Sound on Pi



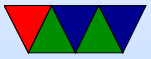
Lab7 – PWM on pico

```
/* set GPIO 28 to be PWM */
gpio_set_function(28,GPIO_FUNC_PWM);

/* get the slice (pwm/channel) */
/* GPIO28 is hooked to 6A */
slice_number=pwm_gpio_to_slice_num(28);

/* The Pico counts at 125MHz */
/* This divides it down so every 255 it wraps to 0 */
/* So roughly 488kHz */
pwm_set_wrap(slice_number,255);
/* This sets A and B channels to go from 0 to 1 */
/* At 128, so 50% duty cycle */
pwm_set_chan_level(slice_number,PWM_CHAN_A,128);
pwm_set_chan_level(slice_number,PWM_CHAN_B,128);
```



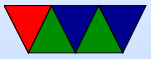
```
/* Finally divide by 255 again which gets roughly */
/* 1419kHz which is roughly B6 on musical scale */
pwm_set_clkdiv_int_frac(slice_number,255,0);
```

TODO: diagrams of the divider/ramps



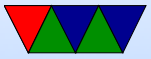
Lab7 – What if want different frequency?

- Can set up a function where you pass in a frequency and it sets up the various values
- Either you can have table of all the notes, or you can calculate. 440Hz is A usually, and 12 notes per octave and you can work it out
- Can have array of notes and lengths and play them in order like a song



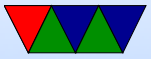
Lab7 – Aside on AY-3-8910 Sound Chip

- 3 square waves with envelopes, noise channel
- Also another envelope
- Every so often (often 50Hz) re-up the values for the square waves
- Found on ZX Spectrum, Atari ST, Apple II Mockingboard



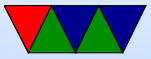
Lab7 – Fitting on Small Computer

- If you want to play back 16-bit audio at 44.1kHz it works out to 5MB/minute
- This is too big for something like the Pico with 264k of RAM!
- Even low quality 8-bit/8kHz is 0.5MB/minute
- This is why compression like mp3 was around, but still too big



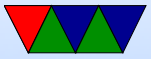
Lab7 – Using a Tracker

- Even simple music, 50Hz times 13 registers is 38k/minute which adds up
- Also writing music with freq/length a pain
- Instead use a "Tracker" where you give it a list of notes, length, and maybe effects
- You can get a player in 2k or so, and songs 2k-8k or so
- Vortextracker, challenges with that



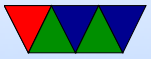
Lab7 – Chiptune Artists

- Amazing musicians
- I am not that skilled. I can enter stuff from sheet music Takes forever, I made a version of Toto's Africa over Winter Break for a project and it took me like a week



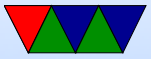
Lab7 – Show off “Pi on Fire Demo”

- Written in C for Raspberry Pi 1B
- Running on custom “vmwOS” from ECE531 class (also written in C) no Linux involved
- Demosplash 2019, came in in second in Modern Demo



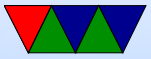
Pointers Redux

- We've talked about pointers but so far they haven't really been useful for much besides strings and also passing things into functions
- It turns out arrays are all secretly pointers too
- The major remaining reason for pointers is dynamic memory allocation



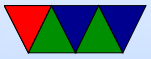
Static Memory Allocation

- So far if we wanted memory, we'd allocate it at compile time
- If we need an integer we `int x;`
- If we need a framebuffer array maybe `char fb[320][240];`
- What if we don't know in advance how much memory we need?



Dynamic Memory Allocation

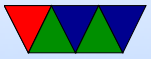
- Wouldn't it be nice if we could only allocate exactly how much memory as we needed?
- Maybe we could also free it up when we were done?
- You can do this in C but it's a lot messier than other languages and also uses pointers



Aside: Variable Length Arrays

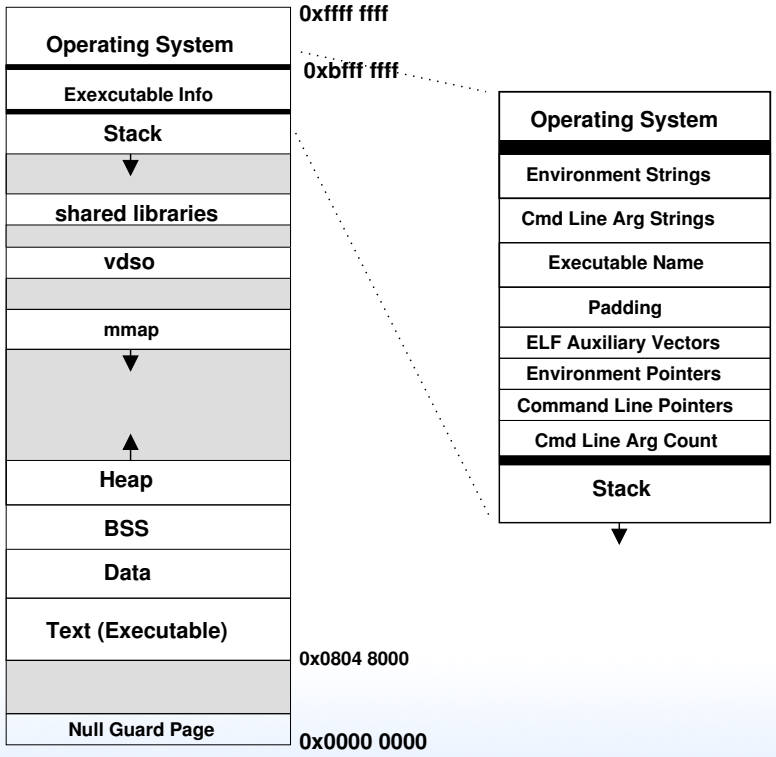
- Maybe we can avoid all this by declaring an array using a variable for he size? `int x[i];`
- Variable Length Arrays (C99) (but made optional in C11)
- Stored on a stack so must happen inside function or block

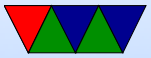
```
void foo(int i){
    int x[i];  // VLA
}
```



aside, reminder on memory layout

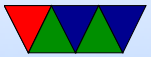
This is on Linux system with virtual memory, going to be a bit different on Pi Pico





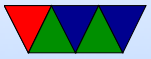
malloc()

- Need to `#include <stdlib.h>`
- `void *malloc(size_t size);`
- Can be used to allocate "size" bytes of memory
- Returns a pointer to the memory that was allocated
- This memory lives in the "heap"
- This is not a syscall, on Linux either `sbrk()` or `mmap()` is used to get the memory



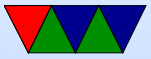
malloc() examples

```
char *bytes;

bytes=malloc(1024); // allocate 1024 bytes
bytes[5]=4; // can treat like array

// in the old days you maybe had to cast
bytes=(char *)malloc(1024);
// but these days it's a void pointer which
// can be assigned to any pointer
// yes, not very type safe

// allocate array of 10 ints
int *array;
array=malloc(10*sizeof(int));
printf("%d\n",array[0]);
```



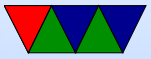
calloc()

- `void *calloc(size_t number, size_t size);`
- size is the size of the objects to allocate, and number is how many
- The main difference, besides the different calling convention, is it zeroes out the memory for you



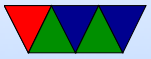
calloc() examples

```
char *bytes;

bytes=calloc(1024,1); // allocate 1024 bytes

// allocate array of 10 ints
int *array;
array=calloc(10,sizeof(int));
printf("%d\n",array[0]);
```

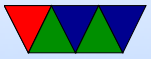


NULL

- Can calling `malloc()` fail?
- Why might it fail (what if not enough RAM?)
- Will return `NULL`, the special value that means invalid pointer
- Should always check for failure. But what can you do if it fails?
- NULL might be 0 but that's not guaranteed so don't trust that
- Note: confusingly this is different than NUL, the 0

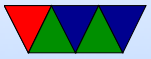


terminator in a string



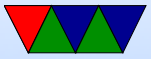
Dereferencing NULL Pointer

- What happens if you dereference a NULL pointer?
  On a machine with memory protection (like Linux) it will crash
  segmentation fault
- On an embedded system like the pico it might not, so be careful
- This is a common cause of security issues



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.



We'll continue with this topic next lecture