Sizeof()

* + **SIZEOF DOES NOT GIVE YOU THE LENGTH OF AN ARRAY.**
  + **YOU CAN'T GET THE LENGTH OF AN ARRAY AT RUNTIME.**
  + sizeof() gives you the NUMBER OF BYTES an array takes up
  + sizeof() NEVER gives you the length of anything at a pointer

**ONLY for array variables:** you can get their address **by using their name alone *or* with the address-of operator.**

* **accessing the value** at an invalid pointer is **undefined behavior.**
  + it might segfault.
  + it might mess up other variables.
  + it might "work" on one computer and crash on another.
  + **you don't know!**
* if you really think about it, "pointing to a single value" is the same as "pointing to an array of length 1"…
* **#define X 10**
  + TEXTUAL replacement of X with 10, everywhere
* **enum { X = 10 };**
  + makes a constant VALUE, but not a variable
  + MUST be **int**!
* **const int X = 10;**
  + makes a VARIABLE (has an ADDRESS)
  + can’t assign const to non-const

**float f = 3.567;**

**int\* p = (int\*)&f;** *// p points to f...*

**printf("%08x\n", \*p);** *// interprets f as an int!*

* this kind of cast does **not** change anything in memory
* **f** is still there and it still holds 3.567
* it only changes how we **view** that memory and "pops" its AR, but since popping is just moving a pointer, the values of x and y and the return address are *still in memory.* *then, we call a second function, and it reuses the space. so its uninitialized local variables a and b take on the values that x and y did in the other function!*

locals are allocated on the stack (within activation records), and their lifetime is the body of the function

**new** marks the beginning of the object's lifetime, which is **independent** of the variable(s) that point to it.

* you can't *access the data* from a void\* without casting
* if you do this (don't do this):

**while(1)**

**malloc(1048576);**

* your program will run out of memory
* but you won't get an error
* this will **loop forever** because if you run out of memory, **MALLOC RETURNS NULL**

when you free heap memory, **all pointers to it become invalid, and it's your responsibility to never use them again**

- here's another kind of invalid pointer: it *used* to point to some valid heap memory, but not anymore.

- just like a pointer to a local that *used* to exist, but doesn't anymore.