9/10 SCRIBE NOTES ----------

Data Representation

C abstract machine:

- defines the meaning of C programming language

Compiler:

- translate C abstract machine to x86 programs
- how to represent any data type?
- related data should be stored together in memory because they will need to be brought to the processor

x86 Processor:

- very fast
- a little dumb
- doesnt understand many types of data and small memory
- defines the meaning of a program (an executable)
- Originally the 8086 processor from Intel
    - All future Intel processors were backwards compatible with this one and had names like 80286, 80386… so we refer to the architecture as x86

Where does the compiler put information?

- How does compiler take all of 'these' variables and put them in memory?
- C abstract machine says: Distinct live variables refer to distinct live objects. Assigning one variable doesn't affect any other.
- Any program that executes an undefined behavior, the compiler is allowed to do anything it wants

How do the compiler and processor represent information as bytes?

- The compiler assigns distinct address in memory to objects
- The x86 processor: Distinct live variables occupy distinct regions of memory

To print an address: printf("%p\n", &local)
&: takes an object and turns it into an address/pointer
don't use on dynamically allocated memory because we already have the pointer

Hexadecimal is noted by 0x_ _ _ _ _ _ _ _ in C
Hexadecimal digits have 4 bits
Binary -> Hex conversion is easy:

- group 4 bits and convert to hex digits

$2^{31} = 10000000000000000000000000000000_2 = 0x80000000_{16}$

***at this point, Kohler drew 31 zeros on the board preceded by a 1 ***

In l03/printvariables.c:

- 48 bytes between function and subfunction
- global and global_array are separated from const_global and const_global_string[0] in memory even though they're declared in sequence in the code
- locals are stored separately in memory

Regions of memory:

- Bottom/beginning: Code and READONLY data (TEXT)
- Then Data
  - Set for all time when program is compiled
- Then Dynamically allocated data (HEAP)
  - Changes constantly during run of the program, grows 'up'
- Then local variables of functions (STACK)
  - Changes constantly during run, grows 'down'
- The rest is the kernel/OS
  - kernel is a program that is always running that makes sure that all other programs always run nicely together
  - Can't be touched

Memory in the stack is constantly randomized to avoid certain types of memory attacks

When a function returns, all of the local variables in that function die and we can reuse the addresses

an int is 4 bytes, but an int pointer is 16 bytes in memory because of metadata added by dynamic memory allocation

```
  0x8048620
- 0x80485f0
------------------
         0x30
```

Heap starts out as empty and inaccessible