**10/14**

Void \_\_free\_input(void \*ptr);

* Isn’t passed the length
* Unmap?
  + Might be slow if unmap is used
* Use some kind of header in memory before the pointer (blank, add size, mmap, next)
  + Size
  + Typedef struct {  
    size\_t len  
    void \*mmap\_start;  
    size\_t mmap\_size;  
    void \*next;  
    } head\_t;
  + Get user pointer, subtract size of header, obtain infos
  + Header\_t \*free\_memory = null
* Start with implementing free
  + Suppose the memory you had is now free
  + Put things in the ll of stuff that is now free
  + Go over list of free blocks (when using malloc)
  + If you reach the end and don’t find end, implement a new block of memory
* Keep track of list by implementing pointers
* Don’t let memory blocks knock into each other
* If memory blocks are 0 blocks apart, combine them into one block
* If entire mmap is free, unmmap it
* When you initialize an mmap, put a header and implement the ll structure
* When you mmap memory (lost cause situation), put this into ll of free blocks. Implement the unmmap stuff….??????????? Or nah?????????
* Check initialization (if initial, do nothing. Otherwise, set up LLs … Check if NULL)
* Don’t delate munmap … Use it as soon as a block becomes free.
* Round **up** all allocated sizes to multiples of the size of your header.
* Go through free memory list, if you cant find the size you are looking for, call some function that creates the amount of space you need in order to create whatever it is you need.
* Cant mash mmaps together.
* Header\_t \*block;  
  void \*ptr;
  + Block + 5; == &(block[5]) //Gives the 5th element of the thing
  + Block +5\*(sizeof(header\_t));
  + Cast pointer to void \*
  + Ptr = (void \*) block;
  + Ptr + 5; -> (Gives the address) 1234+5\*1
  + A pointer is an array

Order of operations

1. Typedef header (contain next and length)
2. Implement free
   * Subtract the header size
   * Insert block into LL (ordered by memory addrs)
   * Colace blocks around newly inserted block
   * \*possibly\* munmap if it fills an mmap block completely (STUB)
3. Implement malloc
   * If size == 0, do some special things
   * Get block from LL
   * If unsuccessful mmap block in
   * Try again
   * Add header size to ptr
4. Implement calloc
   * Multiply some things with your own malloc, slap some zeros somewhere
5. Implement realloc
   * Realloc(old\_ptr, new\_ptr)
   * old\_ptr == NULL, behave like malloc
   * new\_size == 0, behave like free
   * old\_ptr – header size -> gives you old size
   * old\_size > new\_size -> Give back memory
     + Make a block with header out of returned memory and insert into LL
       - Basically, new amount of memory needed is smaller, break it up and slap a header in the new free block (because the memory isn’t needed)
   * Old\_size < new\_size -> Some things
     + Need to give larger memory block
     + Basic implementation: malloc new\_size (with own malloc)
       - Copy contents of old\_block w/o header to new\_block (w/ old size)
       - Free the old\_ptr (with own free)
       - Return new\_block
     + Optimized implementation
       - Before copy, check in free mem list if the block can be extended
   * If old\_size = new\_size
     + Do nothing
     + Return the same pointer