Assignment 1: ++Malloc
CS214: Systems Programming
Section 1
Gene Oh

In this malloc project, a linked list was used to represent the metadata of each memory block. The metadata contents consist of the size of the memory block, the condition of the memory block, and finally the next pointer to the next metadata. Here is a visual representation of what the implementation looks like:

Myblock[4096]

A| 12	B|30	C|50	D|40	E|20	F|11	…	….	…	…

Here, A,B,C..etc represents the metadata block. It takes up 11 bytes of space and the following divider divides the block for the demand in bytes for memory allocation. The total size of bytes equates to the entire block, which is the sizeof(struct block) + numbytes(user asks for). From A to B to C and so forth, there is a pointer linked to each block. When calling the myfree(void ptr) function, there are a few cases we must cover to further avoid memory fragmentation. We will use the example above to cover the various cases.

1st case: if we call myfree on C, we check it's neighbors and see if they are currently being allocated or currently free for merge. Lets claim B and D are free.

A|12 | B|120 +2*sizeof(struct block) |E|20 |F|11 |…. |…. |…

After merging C and D into B, we set B's free flag to 'f', indicating that B is now free for memory allocation and combine 2*sizeof(struct block) and the number of bytes within C and D into B. Within this case, if we were to recall free on C or D, we make a check on whether C and D are currently valid in memory. If it is invalid, we return nothing and claim the ptr is invalid or was never allocated. If however, we free on block B, we also return and claim the block has already been freed.

2nd case: If we call myfree on C and find that B is currently being allocated while D is free, we merge D to C.

A |12 |B| 30 |C|90 + sizeof(struct block) |E|20 | F|11|….|…

3rd case: If we call myfree on C and find that B is free while D is currently being allocated, we merge C to B.

A|12 |B|120 + sizeof(struct block) |E20| F|11| … |…

In the mymalloc function, we first cast a void pointer to the first element of the main block. At the first element of the block, if there is no size then we can claim that no allocation has been set up so we must initialize our first metadata block. While traversing through the list of metadata blocks, if we find a free block which is greater in size than the requested amount, we proceed in splitting the block such that the function returns the exact amount.

A|120| B|200| D|32| …|….

In this example, if the user calls mymalloc(150), we find the a block using the first fit algorithm. Claiming B is free and that B's total size covers 150 + sizeof(struct block), we split block B and return the newly allocated block C. Result:

A|120 | B ||(200-sizeof(struct block) – 150)| C| 150| D|32 |… |…

Realizing that we must coalesce free blocks in a particular fashion was the most interesting finding. For instance, referring back to the 1st case in the myfree function, we cannot merge B and C *into* D because then we are unaccounting for all the space that C and B takes up. It is absolutely necessary to carefully merge always to the left-most block of interest to make sure we're including each block.