Joshua Kim and Hunter Morabito
Andrew Tjang, Ph.D
CS214
Assignment 1: Malloc() and Free()

Implementation

Our malloc works by using 2 bytes for meta data size. Specifically, it uses an unsigned short int for the size and uses the least significant bit (LSB) to use as an allocated flag. If it is allocated the last bit is set to 1, else it is cleared to 0.

There are 3 pointers -- prev, curr, and next -- which help maintain where we are in the 5K block of memory and where the next meta data and data should go. For instance, if a free has been called previously, and a malloc call happens next, it resets the current pointer to the start and traverses the 5k memory block check whether or not previously freed blocks are suitable places to put in the new data from the most recent malloc call.

If a free is never called, it just continues to put meta data and data where the current was updated.

For free, it is ideal to free from the last or tail block since it treats it as if that block was ever allocated in the first place (it resets both the meta data and data to 0 in the myblock[5000]).

Otherwise, it just resets the data to 0 and changes the LSB of the meta data to 0 to indicate that, that block is no longer allocated. (This works hand in hand with the free because if free is called after a malloc call, it traverses and checks that space and sees if it is allocated or not).

For the most part, it will just change the LSB and reset the data blocks to 0. If free was previously called, it will check ONLY the next neighbor. If that block also happens to be unallocated, then it merges it together by updating the metadata size block that is lowest in memory space. The function then updates the data block and meta data block that is higher in memory space to 0.

Possible Errors and Defects

For the most part, our malloc and free implementation works well when malloc and free calls are kept at a minimum. Because we our free doesn't account for the previous unallocated block when merging, it causes fragmentation issues since it is possible that it can look like this in memory:

| FREE (previously freed) | FREE (RECENTLY MERGED WITH → NEXT FREE NEIGHBOR) |

Therefore, when we we running the test cases, we ran into issues, specifically with case D.

These outcomes/results are addressed more indepthly in the testcases.txt

Error outcomes

So, we did return NULL in cases where there was no memory left to allocate with a printf message stating what the error was.

Similarly, we fprintf(stderr, "<Error message %s %i>", file, line); when the free function tries to free a pointer that has not been allocated.

We weren't necessarily sure if we should have text specifying these errors, because the test cases sometimes would output lines of errors.

If is not supposed to, please don’t be alarmed as we have made sure that it does in fact work through gdb.

Thank you & Sincerely,
Joshua Kim and Hunter Morabito