CS214
Assignment 1
Professor Tjang
mymalloc() and myfree()
Eric Zhuang, Rahim

We allocated 5000 bytes of memory for this implementation. We use a doubly-linked list in this implementation, which allows memory to be allocated in linear time O(n). Doubly-linked lists are also more helpful for our myFree() function when attempting to defragment separate sequential blocks of memory, as the array can communicate with previous and next pointers from each node.

When the first call to myMalloc() is made, myMalloc() will initialize a base struct that will initially hold all free memory. A pointer to the address of each memory entry to be allocated will also be saved into an array memEntries (as they are being allocated) to ensure that when a memory entry needs to be free'd, we can check that that specific memory entry was created by myMalloc().

Our implementation detects errors that occur when:

- User tries to free unallocated variables and free'd variables.
- User tries to allocate less than 2 bytes in size.
- User tries to allocate a block of memory larger than what was initially reserved (5000 bytes).
- User tries to free pointers that were not allocated by myMalloc().

myMalloc() will then proceed with a series of checks that compare the size of each memory block in our doubly-linked list and also the amount of memory needed for the malloc request and process with the size of the memory entry from user input.

> **Case 1**: Not enough memory.
>
> **Case 2**: There is enough memory for the malloc request but not enough to make a newmemory entry.
>
> **Case 3**: There is enough memory for both the malloc request and new memory entry.

myFree() communicates with the global memEntries array to check if the requested memory entry freeing operation is indeed valid, and that the specified memory entry was indeed allocated by myMalloc().

myFree() accounts for (de)fragmentation through a series of 4 cases:

Case 1: Prev and Next blocks are free, so both Prev and Next are merged into the now larger current block.

Case 2: Prev is free but Next is not free, Prev is merged into the current block.

Case 3: Next is free but Prev is not free, Next is merged into the current block.

Case 4: Both Prev and Next are not free or there is only 1 memory entry