Vivian Trieu
Kyle Frick

The way our malloc is designed is that we used 4 bytes for metadata, where the first two bytes hold our magic number 5223. When finding a pointer, if we find this magic number, then it is clear that the pointer has been allocated by our malloc. The next two bytes hold the size of the allocated space. When space is allowed, a character '-' takes place of the allocated space. This will indicate that the space is in use. When the space is free, the block will be 0. For freeing, there are a few accounted for cases: if the pointer is out of range of the allocated space, if the pointer is an allocated pointer, if the pointer is in the middle of allocated space (not to an actual pointer), or to a space that was not allocated by malloc (static variable). When freeing, we combine the free spaces together so that there are no fragments of free space.

All workload time is in milliseconds. The times are as follows for 5 runs:

Workload A:

1. 0.78000
2. 0.74000
3. 0.78000
4. 0.68000
5. 0.70000

Workload B:

1. 0.96000
2. 1.03000
3. 0.97000
4. 0.99000
5. 1.01000

Workload C:

1. 144.3600
2. 144.3200
3. 147.7500
4. 144.1000
5.

Workload D:

1. 266.7800
2. 260.6600
3. 272.4100
4. 261.0200
5. 257.5900

Workload E:

1. 223.8200
2. 221.3900
3. 226.6600
4. 222.8200
5. 219.22

Workload F:

1. 743.6100
2. 729.6100
3. 742.8400
4. 741.7200
5. 729.1200

For each workload, each time hovers consistently over an average time. It is clear that under a heavy load such as F, our malloc takes significantly longer to run versus a light load such as A. Overall, the heavier the load gets, the longer it takes for our malloc to run. However, an apparent error that occured when running through the program is that occasionally the values of the time would return negative which may be due to floating point errors in C.