Helen See
HyunMo Yang

Implementation

We chose to implement the algorithms that were presented in class. Specifically, the myFree and myMalloc, with the do while look. With this algorithm, we are allowed a structure for a doubly linked-list which will allow a constant run time. The structure, struct mem, contains two pointer, of type struct mem *, for previous and next. It also contains integers for size and isFree. Size will hold the size that is currently allocated or available. The isFree integer will be either 0, for occupied space, and 1, for available free space.

With this structure, keeping track of allocated space and free space is easy. When free is called, the space pointing at previous and next will help determine whether or not the space needs to be merged with another free space or not.

Currently, the makefile compiles 2 executable. One is for main.c, which can be replaced with a different file but currently has our testcases, having an executable called **mymalloc**. The other one is for our ExtraCreditTestCases.c file which is with the executable called **ecmalloc**.

Extra Credit

After running the makefile, the extra credit test cases is in the executable file: **ecmalloc**
The extra credit we did are:

- Leak detection and reporting (with the extra credit test cases)
- calloc()
- Trying to free() a pointer that is not in the heap

Our calloc function, mycalloc, is almost the same as our mymalloc. The conditions are slightly different. There is the number of items and the size per items. Also, before the allocated space is returned, every spot in the allocated memory space is initialized with zero.

For leak detection and reported, there is a boolean in mymalloc.h and mymalloc.c. The mymalloc.h is extern int forLeaks. In mymalloc.c, forLeaks is int forLeaks=0. By doing this, it will allow a different main file, our extracredittestcase.c, to have test cases requesting for leak information when the main file ends. So, the leak detection and reporting will not run unless the main file changes forLeaks to 1. When forLeaks is equal to 1, then atexit(leakCheck) will be called to run the function leakCheck() at the end.

When the user tries to free a pointer that is not within the heap, an error message will print out indicating that. The code simply checks if the pointer is less than the pointer location at the start of the heap or greater than the location of the end of the heap.