Christopher Wallace caw251
Jerry Zhang jz570

READ ME

Usage
Go to directory containing all the files, mymalloc.c mymalloc.h memgrind.c
```
$ make
$ ./memgrind
```

mymalloc.c

The 'myMalloc' function works to implement first fit. Starting from the head, the program iterates down the list of metadatas by following the next* pointer, checking to see which are free and which have compatible sizes. If a free metadata has more than sufficient size, it will allocate the block that was asked for, and create a new metadata at the tail end, which points to the remainder of free memory until it hits a new metadata, or reaches the end. If the end of a chunk is reached, and no new metadata can be added, the program would throw a warning (commented out).

The 'myFree' function works in a similar matter, iterating down the list of memEntries, starting at head until the address it was sent is either found amongst the list of memEntries, or confirmed to not be a part of the list. If the requested address is NOT found, it throws an error and exists. If the address is found, but the entry is already free, it throws another error. If both conditions are satisfactory, however, it simply sets the 'isFree' variable to 1, allowing another call to simply overwrite the bytes it is holding.
Perhaps the most frustrating aspect of 'myFree', is the block condensing. After a free, the program points to the head node again, and checks if the node and its neighbor are both free, and if so, combines them, this time by simply extending the 'totalSpace' of the first entry to encompass all of its neighbors, and pointing to the neighbor's neighbor instead, effectively removing it from the program. It does not change any of the values in the neighbor, so trying to free it again will still throw the 'this is already free' error rather than the 'this is not an entry' one. Lastly, to prevent any issues involving a loose entry, it checks if the head node is free, and if it has a free neighbor, and will combine them.

mymalloc.h

The header file serves to initialize the myMalloc and myFree functions, for use in memgrind. It also defines the struct that will be used to keep track of free blocks and remaining memory space.

memgrind.c

File made to to test workloads of various sequences of malloc and free calls. Each workload runs 100 times and records the total time and average time taken.

Average Workload times – in the code there were several printf statements for each malloc and free operation – just to let us know that those functions were working as intended. Turns out those prints took up a lot of run time, so once we knew it was working properly, commented them out for a more realistic timing of malloc and free operations.

Data Set - Average Times
Workload A: 5.32 microseconds
Workload B: 51.05 microseconds
Workload C: 4.47 microseconds
Workload D: 6.99 microseconds
Workload E: .63 microseconds
Workload F: 46.129 microseconds

Conclusion:
From the data recorded, Workloads B and F took longer. Workload B was similar to A, except that after freeing immediately after mallocing, it malloc'd 150 times before freeing 150 times. Since A freed immediately, the blocks position in the list did not need to be stored, while for B, it had to iterate through twice, making B much slower.
Workload E has the shortest run time, since it doesn't have iterations, just a handful of pointer mallocs and frees. This workload was to test the capabilities of our mymalloc function. And is described further in the test plan.