Project Assignment 6 Nikolay Feldman Ankur Save

Usage:

Our Test Programs:

Just run make and execute any of the test binary files.

Own Custom program:

Run make and link the source code of a program to the compiled "mymalloc.o" object file. Be sure to include the mymalloc.h header file in your main program to make use of the #definitions for malloc and free.

*NOTE * Max amount of allocatable memory: 25000 bytes*
*NOTE * It is adjustable though from within the source file.*

Description:

This is our implementation of the malloc() and free() library calls. The benefits to our implementation is that it allows us to detect common dynamic memory errors. We use a **static char array** to manage the dynamic memory so that we don't have to worry about where our memory actually comes from.

The **static char array** holds our dynamic memory. There is another **array containing void pointers to memory entries (memEntries)**, the size of *MAX ALLOCATABLE MEMORY* divided by the *SIZE of our memEntry struct + 1*. All cells in the array are initialized to 0 (NULL).The purpose for this is to keep track of **legitimately malloced** memory space and is used in detecting a lot of the errors.

When you malloc(), a block of space is given based on the requested size. Right before the block of space is the MemEntry struct header that lets us handle how much space is used and/or available. Not only is this full block set aside to you and the pointer returned, but the pointer to the MemEntry struct is also stored in the first index of the memEntries array that is NULL (starting from index 0). When you free(), the the size of our MemEntry struct is subtracted from the address contained within the pointer and is checked against our memEntries array for validity. If all is well, the block is freed and merged with nearby free blocks. If the pointer is invalid, an error is returned.

Errors that are caught:

- Freeing NULL pointers
- Freeing pointers that were never allocated
- Freeing pointers to an address NOT returned from malloc()
- Freeing pointers redundantly (freeing when already freed.)
- Allocating for a block the size of 0 bytes
- Saturation - Allocating for a block when not enough memory space available
- Fragmentation – The fix is to increase maximum amount of dynamic memory available

Efficiency:

Assuming the size of our allocatable memory space is **n**, then we have **O(n)** for memory space.
Assuming the # of malloced entries is **m,** the runtime efficiency of **malloc()** is **O(m)**
Assuming the # of malloced entries is **m,** the runtime efficiency of **free()** is **O(m)**