Christopher & Sandeep
Malloc && Free

Variables:
-We used a static MemEntry * to keep track of all the memEntries
-We used a static char [] to allocate and give out memory to users who ask for it
-We used two static size_t variables to document the amount of memory given out and the number of MemEntry structs that were free
MemEntry is a struct used for book keeping
MemEntry is the ultimate book keeper
-we use it to determine how many things we currently have stored
-we use it to determine if the data has been free

Implementation:
MALLOC
-To keep track of amount of data given out to end users, we created a struct called MemEntry to store size of data given out, whether or not the struct is free, a code to verify the MemEntry struct is valid, and pointers to the next and previous MemEntry struct
-When a user asks for data space the request goes the following checks
1)the user is asking for a valid amount of data
2)malloc has enough space available
3)if malloc doesn't have enough space and has free memEntry structs then we attempt to give the user a pointer to the data directly after the memEntry struct
-When we use an exisiting memEntry struct (this only happens if the user is asking for more data than malloc has free) then we attempt to slice the struct to only give the user a pointer to the data requested. If this is not possible, then the slice fails and just return a pointer to more data than user requested
-if the user has not requested any data space yet then we create a memEntry struct and then return a pointer to the data after it
-Else we just append to the end of the data

FREE
-We look to make sure ptr being freed is valid
-If invalid, return an appropriate error message
-Most of the logic for the free is found in defragment which does the following:
1)Determines the memEntry * position in the list
2)If memEntry * is at end of list, then delete and adjust memAllocated variable
3)Else then look at the memEntry before and after memEntry * to be freed to see if any memEntry * can be combined to create a single memEntry *
-An interesting thing about our free implementation is that our free only checks the memEntry * before and after the memEntry * to be freed, that way we don't traverse the whole list every call to free

CHECK IF INVALID POINTER
-Before we free the pointer, we move it back the size of a MemEntry struct, that way we can edit MemEntry
-Before we start editing MemEntry, we check if we are trying to access a valid memory location
-If invalid memory location, then we return 0;
-If there is no MemEntry, return 0;

TESTS:
-Test0.c covers all the basic freeing and malloc
-Test1-6 handle more special cases
-Test 7 covers everything the professor mentioned in the asst detail