readme.pdf for mymalloc.c

Essential components of our better memory allocator:

MemEntry Struct

unsigned int specialcode: this is used to determine if a pointer is pointing to a MemEntry struct.
size_t numbytes: denotes the numbytes that are being reserved by a MemEntry (these bytes may be free or not free).
int isfree: denotes whether or not the space referenced by a MemEntry is free
int arrayno: denotes which 5000 block of contiguous memory the MemEntry is located in (this was added for when we use sbrk to provide the caller with more memspace...each 5000 block may not be contigous when returned from sbrk).
MemEntry * next: points to the next MemEntry (usually after space requested)
MemEntry * previous: points to previous MemEntry

---

mymalloc function:

mymalloc's essential function is simulate how memory gets allocated through the standard malloc(). This simulated malloc takes a size_t value (because it is impossible to have negative memory) and adds a MemEntry struct to the memspace array that will be used to bookkeep the space requested by the user. Mymalloc then returns a pointer to the space requested for use by the caller. A simple diagram below denotes the basic structure of the array after a few entries have been made.

Mem Entry	Space Requested	Mem Entry	Space Requested	Mem Entry	Space Requested	Etc..

mymalloc.c begins by accepting a size_t var which is the number of bytes that needs to be allocated and then checks to see if there is a MemEntry at the beginning of the memspace array

If there is a MemEntry at the beginning of memspace (occurs whenever it is not the first call to mymalloc) mymalloc enters a while loop and will iterate until it reaches null or until it finds a free MemEntry struct with enough numbytes referenced to store the requested size by the caller. If there is enough free space referenced by an already existing MemEntry struct there are two possibilities:

1. There is enough free space for the space requested by the caller and a new MemEntry struct plus at least one free byte.
2. There is enough free space for the space requested by the caller but not enough space for a new MemEntry struct plus as least one free byte.

On the event that 1 happens, the free space referenced by the MemEntry will be “split” so that the current MemEntry will be adjusted to the exact size requested by the caller and the remaining space will be referenced by a new – free – MemEntry. This fragments our memspace array for more efficient data storage.

On the event that 2 happens, the MemEntry will simply be switched from free to not free and a reference to the space requested will be returned to the caller. In this case the user may receive a larger chunk of data than they asked for...however since they should not be going out of bounds regardless they will never notice this. This was done for efficiency so that space referenced by a MemEntry can be re-used without necessarily having to be “split.”

If there is no MemEntry struct at the beginning of the memspace array (that is, this is the first time mymalloc is being called) a MemEntry struct will be placed at the beginning of memspace referencing the space requested by the user, and another MemEntry struct willl be added immediately following the first one (as long as there is space for it) referencing the remaining total space in the memspace array.

EXTRA CREDIT 1

This program uses sbrk to allocate more memory if needed. If the the size being requested is bigger than the number of bytes free at the time then sbrk is used to link another block of memory equal to the original capacity of memspace (5000) to the end of the current memspace array. Since the blocks of memory returned by sbrk may not be contiguous we have only accounted for desiring more TOTAL space than the original 5000 bytes...NOT for space requests greater than 5000. Also, since in our myfree function we will merge together two (or three) adjacent free MemEntry structs (to handle data fragmentation), we included an arrayno field in the MemEntry struct which will keep track of which 5000 byte data block the MemEntry is in. Two MemEntry structs will only be merged if they are located in the same 5000 byte contiguous block of memory.

---

mycalloc (EXTRA CREDIT 2)

mycalloc's function is to simulate the standard calloc() function which is to simply set each byte of the newly allocated memory to a value of 0. This is done by mycalloc using the mymalloc function and then using the memspace function on the pointer returned by mymalloc to the space originally requested by the caller. Memset takes a pointer, a value and a size. By using memset on the space requested by the caller we can ensure that all of the data requested will be initialized to 0 value.

---

myfree

myfree's function is to simulate the standard free() function that is used to free memory allocated by malloc.

This function accepts a void pointer which should point to the beginning of the space that was originally requested by some other call to mymalloc. To ensure that this pointer is valid, it is decremented the length of one MemEntry. If it is valid, then at this location we will find our unsigned int specialcode. If it is not valid, we will not find this code. To check this, the pointer is cast to an (unsigned int *) type and checked against the CODE that it should be. If it is not valid an error is returned.

If it is valid, my free will first check to see if the previous MemEntry struct in the list (if it exists) is also free. If it is free, it will merge the current MemEntry with the previous one. It will then check the same for the next MemEntry in the list. Thus, in one myfree call, the freed MemEntry can either be merged with its previous, with its next, with both of these, or with neither. This is done to handle data fragmentation.