README ---------

In order to simulate the heap, we used an array of size 4096
We used a linked list of metadatas where each metadata node contains the following information:

size_t Freeflag -- 0 or 1
size_t sizeContained -- size of the block that corresponds to the metadata node.
next -- contains ptr to the next metadata or NULL if it is the last node in the list.

we used size_t for the fields because it is unsigned so it use less amount of memory which will make the implementation more memory efficient.

mymalloc implementation using first-fit algo:

First check if requested bytes is < 0 or > than 4096. If it is, throw error and return NULL because our simulated heap cannot afford to give negative amount of memory or more than 4096 bytes.

Then we checked if this is the first time the user called malloc. If it is, then we need to create the first metadata node. This only happens one time.

Now the actual algorithm begins. We use a helper method to find a valid accepting Node that can allocate the number of bytes the user requested.

we do this by traversing through the entire list (starting at the head) and looking for a node that has freeflag of 1 and has at least the same size of space that the user requested for. If there is no valid node found, we return NULL.

Now that we have a valid node we need to check the metadata information to see if the space contained is EXACTLY enough or if it has more than enough.

1) If it has exactly enough the next steps are simple:
set freeflag to 0
we increment the currnode by doing something like ++curr to now point to the beginning of the allocated memory.
Now all we have to do is return that void pointer

2) Else In the case where there was more than enough space:
we need to allocate the memory that the user asked, return the proper void pointer AND ALSO make a new meta data containing the information about the rest of the block that was not needed.

3) If it was neither of these cases then there was no valid node that could accept the amount of bytes the user requested so return NULL and print error.

myfree implementation:

There a few things you need to check for.

1) Check if the pointer given actaully occurs within the address range of our simulated heap. If it does not, print error and return.
2)If it is a valid pointer then we just need to access meta data by doing --curr where curr is the pointer passed into the free function by the user.
Next we can update the metadata by changing the free flag to 1
If free flag was already 1 then that means this pointer was already free! So we return error because you cannot free an already freed pointer.
However if we free many pointers and there are two free metadatas consecutively, then we are wasting space and we need to merge these two nodes into one and sum up their sizes to one total size contained by a single node.
This is done by traversing the whole list and looking for any nodes that have freeflag = 1 and are right next to each other. If they are, we need to merge them by deleting the right most node and adding its size to the left node.
3)You cannot free a NULL pointer.