Liam King & Mohit Pavecha
Asst1: To Build A Better Malloc

Program Description:

**Design:**
In mymalloc.c there is a mymalloc() function that does the work of malloc(). There is initialize() which initializes the memory space and then allocateBlock() which creates an unallocated block that covers the entire memory array. Then there is myFree() which is our version of free() which uses isFreeBlockBefore and isFreeBlockBehind which check to see if the current block has an unallocated block before it.
In memgrind.c we implemented functions testA(), testB(), testC(), testD(), testE(), testF(), and rand_lim(). The test functions run their respective workloads for one iteration. The rand_lim() function takes two ints as arguments and returns a random number between the two int arguments. In the main function, a while loop iterates 100 times, calling each of the test functions once and adding up the returned run times for each workload. After the loop finished, each summed run time is divided by 100 to get the average run time.

**Workload Data:**
Through a number of test runs, we found the average run times for each workload to be as follows:
A: 7-10 microseconds
B: 272-285 microseconds
C: 47-60 microseconds
D: 50-70 microseconds
E: 270-300 microseconds
F: 25-30 microseconds
We found the run times of workloads B and E to be rather surprising. Both of these workloads run two while loops, the first populating a 150 size array with pointers and the second freeing every pointer in the array. The fact that these workloads ran so much slower than C and D is interesting as C and D also involve populating and freeing every index of a size 150 array but run in roughly a quarter of the time. As they only have one loop, it's rather unsurprising that A and F were the fastest workloads. We also thought that C and D would take much longer than they did as statistically there should have been several iterations where nothing was freed or malloced which should've slowed down the run time a good amount.