For the design of the headers, we decided to use a block structure with attributes size, assigned (char either 0 or 1), next, prev, and address. The total size of a header is 32 bytes. This can create a lot of problems because its quite large for a header. Considering something simple like 150 one byte mallocs, this will create 150 bytes of data and 4800 bytes of header information, barely fitting inside main memory and making the storage useful for large data only. To reduce this size in the future, we could remove prev and address to reduce the size of the header by 16 bytes which would be a good amount. We could also remove pointers altogether which reduce the 32 bytes to 8 bytes total, and we would need instead of using pointers use size as the amount to go forward in the blocks.

The overall design of malloc is as follows:

1. Caste mainmemory to block and check if the address stored in that block is the same as the address of the memory address one block size away from the pointer (ie the start of that blocks memory). This indicates that the memory has been initialized; if not, initialize the memory by creating one block of size 5000 – sizeof(block).
2. Loop through block linked list trying to find blocks that are not assigned data yet and are of a size greater than or equal to the requested size. If found, set firstFree pointer to that address and break from loop.
    1. If no blocks found that are big enough, print an error for allocation size exceeding memory.
3. If the block has just enough room to fit the requested data size or if another header cannot be created when splitting blocks due to data size insufficiency, simply return the address for the data block giving the extra bytes at the end.
4. If you can split to make another header with data size of at least 1, then do a normal split of blocks by creating a new unassigned block from the end of the size of the nextFree block data set. Return nextFree's address.

The overall design of free is as follows:

1. Caste mainmemory as a block and check if mainmemory is initialized the same way as done in 1 of malloc design.
2. Loop through block linked list comparing the address of each block to the requested address. If it does not match, continue. If it matches, check if it is assigned yet. If it is not assigned print error for freeing memory that is already free. Otherwise, set assigned to 0, get the next and previous blocks, if the next block is not assigned, merge the next block into the freed block and update freed size accordingly. If the previous block is not assigned and update the freed block size and merge prev block into freed block.
3. Return
4. If no blocks found matching address, print error with freeing memory not allocated by malloc.

For the typical runtime, we found that because of the way we use a doubly linked list and store a large amount of header information, our program runs in a very quick manner. The following is a typical result:

Average time to run workload C: 0.000028 seconds
Average time to run workload D: 0.000028 seconds
Average time to run workload E: 0.000528 seconds
Average time to run workload F: 0.000394 seconds
Average time for workloads: 0.001049 seconds
Total run time: 0.104891 seconds

I think that if we would reduce the header size, we may increase the run time slightly as there would need to be more looping to find previous blocks (instance x3) and when checking initialization (every time).