Michael Saunders and Sakib Rasul
Assign1
Systems Programming Fa2019 Rutgers University

First of all, we messed around with the different options for storing our metadata.
We learned of the bitfield functionality in C, and decided to use that as we believe it's as small as we can get.

```
mbs189@ls:test$ ./a.out
size of int: 4
size of short: 2
size of unsigned int: 4
size of unsigned short: 2
sizeof "flags": 1
size of bitfield_test: 2
```

"bitfield_test" wound up being the template for our metadata struct.

This was our design and strategy. Although, honestly it was a just a rough idea and the notes we took along the way. The following was written by Sakib as he implemented mymalloc() and myfree().

```
1. Initialize char myblock[4096]. Zero it. This block is the space we'll be
managing.
2. Initialize a metadata struct inside this block that contains fields:
   - size  – the amount of reservable memory
   - in_use – whether or not the corresponding block of memory is reserved
   - prev – a pointer to the previous metadata struct or NULL
3. Say a user asks for x bytes:
   1. Check the in_use field first block of memory. If it is in use, skip to
the next block of memory (if it exists) and try again.
   2. If you find an unused block, compare its size to x:
      - If the block is too small, try the next block (if it exists).
      - If the block is too large, allocate the chunk you need and initialize
the rest.
      - If the block is just right, allocate it.
4. Say a user asks to free a block of memory at address y.
   1. Start at address 0.
   2. Move forward the sizeof(metadata). If y is not equivalent to the
current address, move forward the size of the metadata's corresponding block,
and try again.
   3. If address y is found, then free its block, and concatenate it with any
adjacent free blocks:
      1. Go backwards sizeof(metadata) indices.
      2. Set in_use to 0.
      3. Go backwards prev_size + sizeof(metadata) indices.
      4. If in_use is 0, then set prev_free to 1.
      5. Go forwards sizeof(metadata) + prev_size + sizeof(metadata) + size.
      6. If in_use is 0, then set next_free to 1.
      7a. If prev_free is 1 and next_free is 0, then set the size of the
previous block to prev_size + size.
```

```
     7b. If prev_free is 1 and next_free is 1, then set the size of the
previous block to prev_size + size + next_size.
     7c. If prev_free is 0 and next_free is 0, then do nothing.
     7d. If prev_free is 0 and next_free is 1, then set the size of the
current block to size + next_size.
```

Our results are pasted below for three test runs.
We were expecting Workloads C D and E to be much slower than A and B. That was not the case.

```
mbs189@ls:assign1$ ./memgrind
Mean Runtimes:
        workload A ---- 3.110000
        workload B ---- 24.030000
        workload C ---- 4.850000
        workload D ---- 3.520000
        workload E ---- 1.680000
        workload F ---- 0.040000
mbs189@ls:assign1$ ./memgrind
Mean Runtimes:
        workload A ---- 13.630000
        workload B ---- 107.800000
        workload C ---- 18.300000
        workload D ---- 15.320000
        workload E ---- 8.130000
        workload F ---- 0.150000
mbs189@ls:assign1$ ./memgrind
Mean Runtimes:
        workload A ---- 13.710000
        workload B ---- 110.020000
        workload C ---- 21.860000
        workload D ---- 20.810000
        workload E ---- 7.870000
        workload F ---- 0.090000
mbs189@ls:assign1$ _
```