Assignment 1: Memory Allocation++ with Patrick McGrath and Seung Jae Lim 임승재

Our malloc function stimulates memory allocations using node pointers within the block of memory. For every node that is created, another node is created after with the size equal to the remaining unallocated space within the block of memory. Memory that is freed can be reused (obviously) upon further calls if the size permits. And the free function marks existing nodes as free and merges adjacent nodes, if available, into one larger node, so long as they are both free. Hope you enjoy our wonderful code and runtimes. Below is our expected results. Due to random choosing between malloc and free in few testcases, the time taken may vary 1~6 microseconds.

./memgrind

1000 separate malloc()s of 1 byte, then free() the 1000 1 byte pointers one by one.

Time took for test case A: 54 microseconds

Running malloc of one byte then immediately free it. 1000 times.

Time took for test case B: 51 microseconds

Randomly choose between a 1 byte malloc() or free()ing a 1 byte pointer - do this 1000 times.
- Keep track of each operation so that you eventually malloc() 1000 times.
- Keep track of each operation so that you eventaully free() all pointers
There are leftover 49 stuff. So I'm gonna free them.

Time took for test case C: 305 microseconds

Randomly choosing between randomly-sized malloc() or free()ing a pointer.
- Keep track of each malloc so all mallocs do not exceed total memory capacity.
- Keep track of each operation so that you eventually malloc() 1000 times.
- Keep track of each operation so that you eventually free() all pointers.
- Choose a random allocation size between 1 and 64 bytes.
There are leftover 16 stuff. So I'm gonna free them.

Time took for test case D: 294 microseconds

Malloc an incrementing size. When reached capacity, free all BACKWARD.

Time took for test case E: 10 microseconds

Malloc an decrementing size. When reached capacity, free all FORWARD.

Time took for test case F: 3 microseconds