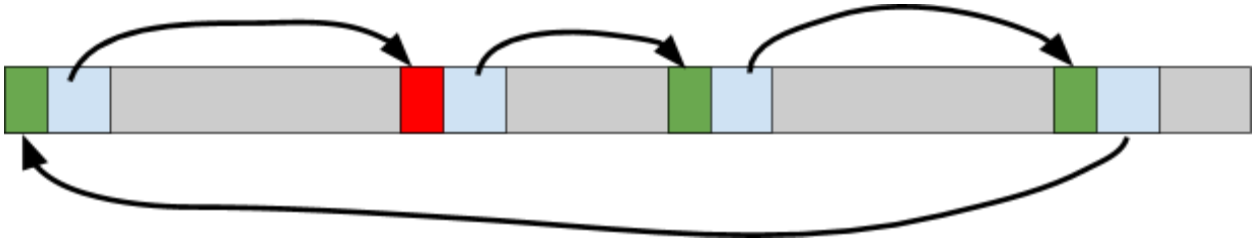
Assignment 1: ++Malloc
Muhmmad Choudhary (mzc13 section 7)
Niranjana Ganesh (ng460 section 4)

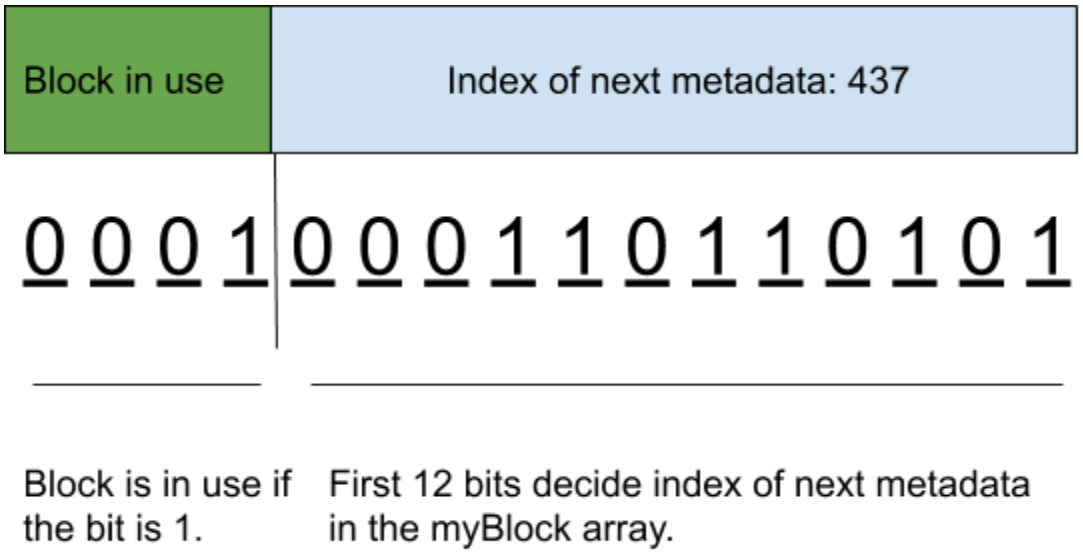
myBlock Design

Our implementation of malloc uses a “pseudo” linked list where each block stores information about whether the block is occupied, and the location of the next block. It is a circular linked list, so the final block will point back to the first index of the array.



Metadata Implementation

Each block uses a 2 byte short int as the metadata. The first 12 least significant bits hold the address of the next metadata, and the 13th least significant bit decides whether the block is in use or not.



mymalloc Design

If the first two bytes of the array are zero, it is as if this is the first time malloc is run. In that case, you can malloc the requested space and if there is enough space left for another allocation, initialize another block that can be used.

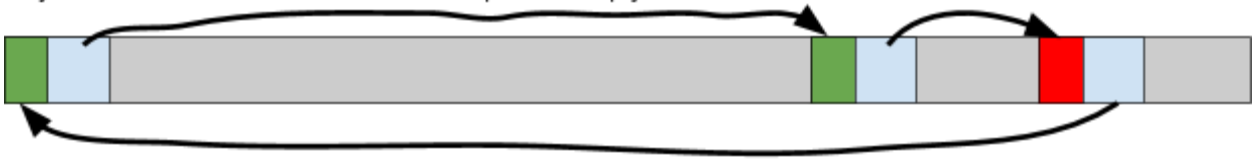
myBlock before first malloc run:

myBlock after first malloc run where entire array is used by one allocation:

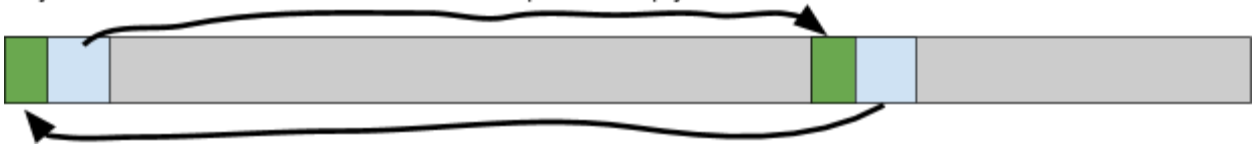
myBlock after first malloc run where there is space for another allocation

On subsequent malloc runs the first unused block that can accomodate the allocation size is either devoted entirely to the allocation or split up.

myBlock after malloc run where there is space in empty block for another allocation

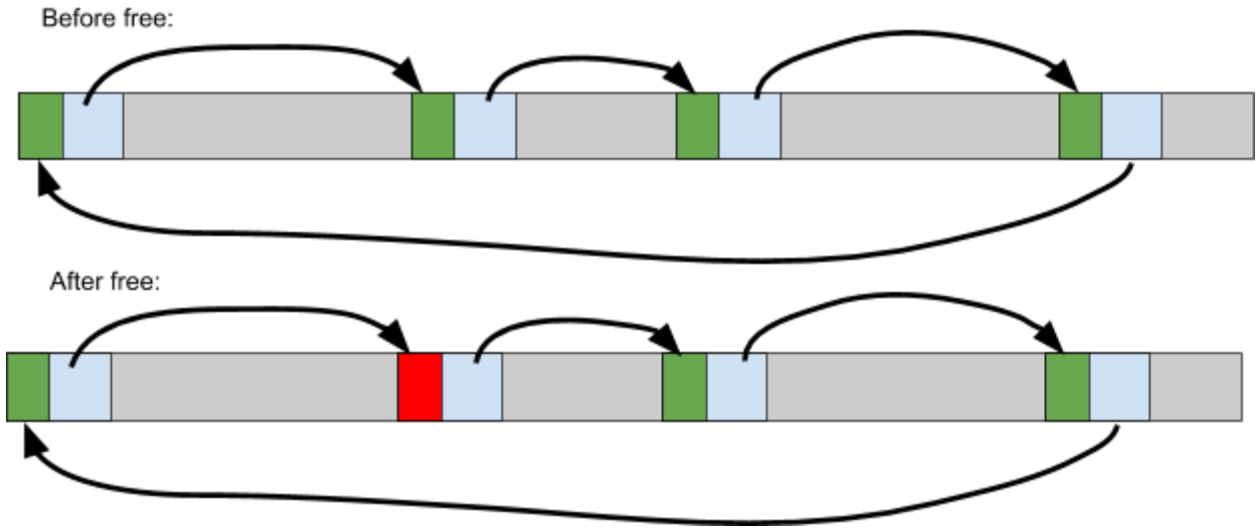


myBlock after malloc run where there is not space in empty block for another allocation

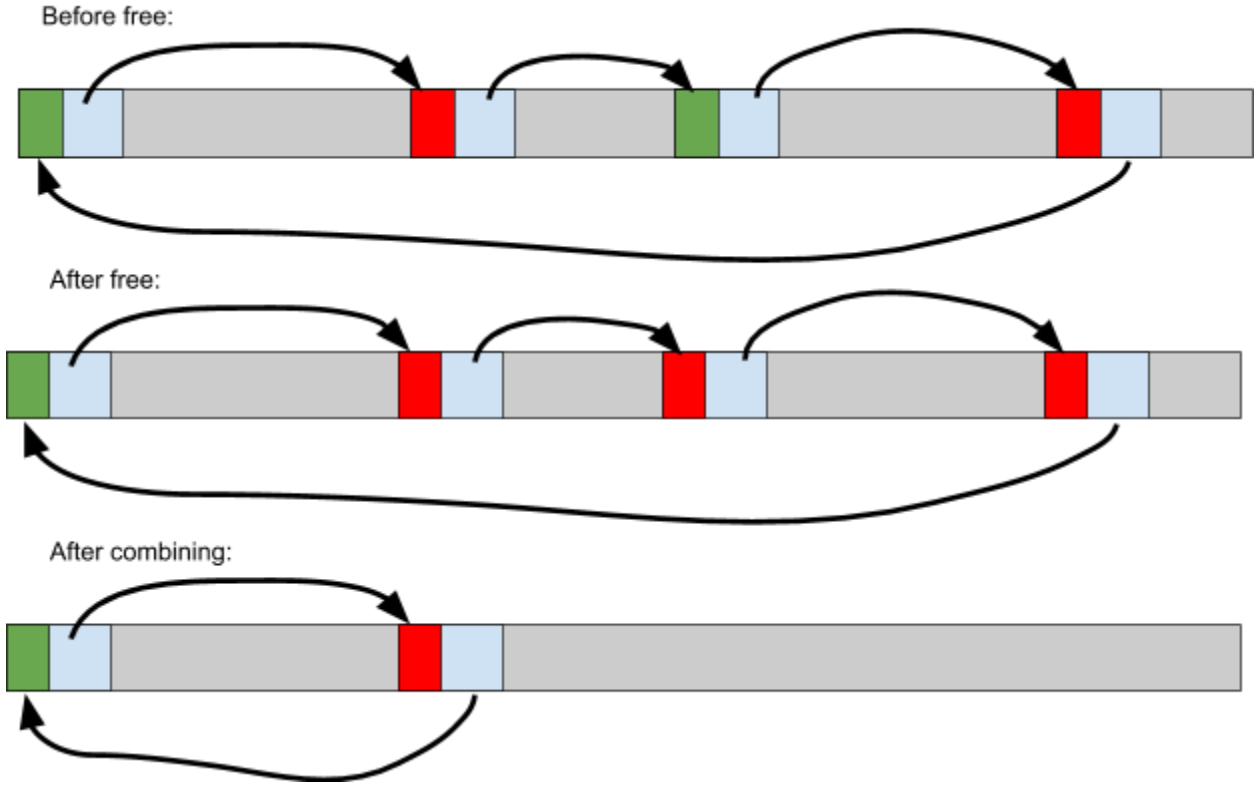


myFree Design

Our myFree implementation traverses through myBlock, checking each metadata, searching for the pointer provided to it. If the pointer isn't found, or has been freed before, it prints an error.



If a chain of free blocks is formed after myFree frees the block, all the blocks are combined into one:



Workload Data

After running memgrind, we get these results:

Test Case A Average Runtime:
2 microseconds
Test Case B Average Runtime:
20 microseconds
Test Case C Average Runtime:
4 microseconds
Test Case D Average Runtime:
5 microseconds
Test Case E Average Runtime:
14332 microseconds
Test Case F Average Runtime:
41587 microseconds

The reason our test cases E and F run so long is because they involve filling the entire myBlock array with allocations. Test Case A is incredibly fast, often times its average is only one microsecond. Timing the functions properly was an unexpected challenge. It took a while to realize that the gettimeofday function wasn't in <time.h> but instead it was in <sys/time.h>. Also, there was an error where our runtimes for test cases E and F would sometimes be negative values. We eventually realized that test cases E and F were running too long to time like the other test cases. For these cases we would not only need to consider the microseconds, but also the seconds value returned by the gettimeofday function. After implementing this change in our code, we started getting precise and accurate runtimes.