Christopher LoPresti

Roslan Arslanouk

CS214

Asst 1: Making a Better Malloc

This program is made to simulate malloc() and free() calls in c while using a static character array (myblock[5000]) to simulate our dynamic memory. At the start of our array, we had a short * keeping track of the available free space in the memory. This lets the memory keep track of its total space, which may determine if we can malloc() or not. Every pointer that we malloc() will point to an address in the array myblock, simulating memory. One index prior to the returned index of the array (pointer) is that respective memory's metadata. The metadata consists of one single character, '~' or '|'. '~' represents that the memory to the right is taken up until the next found '~', '|', or end of memory. '|' represents that the memory to the right of it is free until the next '~' or end of memory. Because we use '~' and '|', we urge you to not use those characters when you initialize characters or strings. For safety, do not initialize an int, double, short, etc., with the values 174 or 176. These are the decimal representation of '~' and '|', and may cause an issue.

The program operates using the first fit method when finding memory. It looks for an '|', and when found it will move to the next '~' (or end of the memory) until it finds a chunk of memory that fits the given size we are trying to malloc(). Once found, it will switch the '|' to a '~' and the memory will now be taken. If we requested a smaller amount of memory then the first free chunk holds, we will split the chunk of memory if it makes sense to do so (if we have enough space left over for an '|' and memory for it to represent). If splitting the memory results in just one singular index that can only hold metadata, then there is no point of doing so, and we just give the pointer some extra memory. If we have enough free space, but no contiguous chunk of memory that fits what we are trying to malloc(), then we cannot execute our malloc() call, and we return NULL. This does run us into an issue, because if you have a pointer that is allocated, and try to malloc() it again, and it fails, the pointer will now be NULL. I wish we could fix this by keeping the address of the pointer the same instead of NULL, however, the definition given for malloc() in the assignment does not allow for that.

When we free() a pointer, we make sure that it is indeed a pointer that was created by our malloc(), and we make sure that the metadata correlating to the memory is taken; if it is free, then we can't free something that is already free. Once we free memory, we look to the left and right, trying to merge with other free memory. If we cannot merge, we just leave the metadata as '|'.

With our own implemented work load we decided to try two similar test cases. In the first test case we pick a random number between 2 and 64 then malloc() with that size as many times as we can. From there we free every other pointer, and malloc() the size-1 (which is why our range is 2-64, as we cannot malloc() 0 bytes) as many times as we can. This will give every new smaller block of memory one extra byte then intended, because splitting the memory with 1 free block as the remainder makes no sense. So if we malloc()ed 15 bytes as many times as we could, free()d every other node, then malloc()ed 14 bytes as many times as we could, we would actually just be getting pointers to 15 byte chunks of memory. We did this case to check our splitting functionality.

With our second test case, we did something similar. We followed the same procedure, except instead of re-malloc()ing the size -1 , we re-malloc()ed the size -5. This now allowed the splitting functionality to work properly, creating chunks of free space that we did not need for our allocation. For this to work, we changed our range of random numbers to be 10-64.

Our findings conclude, that because we had our metadata represented as one character, and that we did not use any structs, we could allocate much more memory than had we used a linked list as our metadata. Each allocation takes at most the size of memory plus or minus one byte for metadata. This allowed us to be much more efficient with our memory usage. Because our memory size was only 5000, our program executed our test cases very quickly.

Example runs for our error checks and test cases A through F (where E is the first text case described above and F is the second, and A- D being in the assignment description) with run time in seconds included can be found on the next page.

```
----------------------------------------------------------------------------------------------------------------

error check one
Error: Memory was not allocated for the pointer in memgrind.c on line: 64, so it can not be freed
error check one time: 0.000003000000000

error check two
Error: Memory was not allocated for the pointer in memgrind.c on line: 76, so it can not be freed
error check two time: 0.000003000000000

error check three
Can not allocate 5000 bytes in FILE: 'memgrind.c' on LINE: '89'
Can not allocate 4999 bytes in FILE: 'memgrind.c' on LINE: '91'
error check three time: 0.000003000000000

Done with test cases
A: average time for 100 itterations took: 0.0000039000 seconds
B: average time for 100 itterations took: 0.0019006100 seconds
C: average time for 100 itterations took: 0.0004981900 seconds
D: average time for 100 itterations took: 0.0007026100 seconds
E: average time for 100 itterations took: 0.0025818800 seconds
F: average time for 100 itterations took: 0.0052802400 seconds

The total time for all the test cases was: 1.0967430000 seconds

----------------------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------------------------

error check one
Error: Memory was not allocated for the pointer in memgrind.c on line: 64, so it can not be freed
error check one time: 0.000003000000000

error check two
Error: Memory was not allocated for the pointer in memgrind.c on line: 76, so it can not be freed
error check two time: 0.000003000000000

error check three
Can not allocate 5000 bytes in FILE: 'memgrind.c' on LINE: '89'
Can not allocate 4999 bytes in FILE: 'memgrind.c' on LINE: '91'
error check three time: 0.000003000000000

Done with test cases
A: average time for 100 itterations took: 0.0000036400 seconds
B: average time for 100 itterations took: 0.0017848000 seconds
C: average time for 100 itterations took: 0.0004761300 seconds
D: average time for 100 itterations took: 0.0006489800 seconds
E: average time for 100 itterations took: 0.0027717600 seconds
F: average time for 100 itterations took: 0.0078965300 seconds

The total time for all the test cases was: 1.3581840000 seconds

----------------------------------------------------------------------------------------------------------------
```