ReadMe
CS214
Shengtao Lin (sl1377) Yang Bao (yb184)
Systems Programming Assignment 1: ++Malloc

Brief Description and Algorithm:

We used a static char array of size 4096 to simulate the computer memory we are allowed to use. Each data that dynamically malloced into the memory will have its metadata and space it required. Our metadata has 5 bytes, the first byte is a char that contains a flag. There are two kinds of flags: 'a' as available and 'o' as occupied. The flag followed by another 4 bytes that we cast it into an integer to store the space/bytes it occupied or available space/byte from this flag to the next occupied flag. By reading this number, we can jump between flags using the pointer. Our program can also respond to detected errors and out in which files and which lines that went wrong. (Improvement: we can short instead of integer since we only need 0~4095 and short is 2^16 which is enough for the space we need. If we use short, the metadata can reduce to 3 bytes. However, we realized that after we finish all the coding and we do have time to do all the calculations about jumping between flags again.)

Malloc:

In the malloc, we accept a size_t size and use traverse to find a space of bytes that can fit that size in. If space is enough, we will use split to change the flag from 'a' to ‘o’, calculate the space is used and recalculate the space available. Otherwise, we will jump to the next available space and check again. If it reaches the end of the array, we will return NULL and tell the user there is not enough space to store that.

Free:

In free function, we will start with the input pointer and three other pointers that can record the previous, next and current flags. It will go through the array and find the address that matches the input pointer. Then it will recalculate the space available and make adjacent available space as one. If the input pointer address is not in the array or it points to a flag that is available, it will catch the error.

Runtime:

Here is our run time result. We find that if the workload implemented with random (Workload C, D, E), the time it take to malloc and free can much less than directly malloc and free.
However, if we use random, the time between multiple trials can be very, instead, the time workload A, B, F take are more stable.

```
[sl1377@less pa1]$ make
gcc -o memgrind memgrind.c mymalloc.c mymalloc.h
[sl1377@less pa1]$ ./memgrind
The average time of workload A is : 330
The average time of workload B is : 33
The average time of workload C is : 3
The average time of workload D is : 331
The average time of workload E is : 34
The average time of workload F is : 2660
[sl1377@less pa1]$ ./memgrind
The average time of workload A is : 330
The average time of workload B is : 32
The average time of workload C is : 3
The average time of workload D is : 3
The average time of workload E is : 31
The average time of workload F is : 2525
[sl1377@less pa1]$ ./memgrind
The average time of workload A is : 330
The average time of workload B is : 31
The average time of workload C is : 3
The average time of workload D is : 330
The average time of workload E is : 31
The average time of workload F is : 2473
[sl1377@less pa1]$ ./memgrind
The average time of workload A is : 330
The average time of workload B is : 32
The average time of workload C is : 3
The average time of workload D is : 330
The average time of workload E is : 31
The average time of workload F is : 2477
[sl1377@less pa1]$ ./memgrind
The average time of workload A is : 330
The average time of workload B is : 31
The average time of workload C is : 3
The average time of workload D is : 3
The average time of workload E is : 31
The average time of workload F is : 2472
[sl1377@less pa1]$ ./memgrind
The average time of workload A is : 330
The average time of workload B is : 33
The average time of workload C is : 3
The average time of workload D is : 331
The average time of workload E is : 32
The average time of workload F is : 2479
[sl1377@less pa1]$ make clean
rm -f memgrind
[sl1377@less pa1]$
```