Homework 7: DIY Memory Allocator

User programs typically allocate memory using the standard library functions `malloc()` and `free()`. These are userspace functions, which both use the `sbrk()` system call to request and release memory allocated to the process.

In this homework, we create an alternative memory allocator, which provides the functions `myalloc()`, and `myfree()`. The performance of your allocator will be measured in terms of program runtime and fragmentation.

NOTE: The template makefile tells gcc to compile in 32-bit mode, even if you're on a 64-bit system. If you get a complaint about missing stubs-32.h, just run `sudo apt-get install libc6-dev-i386`.

Turn-in requirements: make your changes to hw7.c only.Your solution should use only `sbrk()` to request and release memory. This means you cannot use other built-in memory allocation functions like `malloc()`, `calloc()` or `mmap()`. Your solution will be graded based on performance in a variety of scenarios, some or all of which are already present in main.c.

BASIC MYMALLOC() AND MYFREE()

Based on the homework template, support the re-use of free()'d allocations in subsequent calls to mymalloc(). You may want to use a linked list to represent free allocations here, as we don't worry too much about performance.

Like in malloc, make sure you use the free memory to represent the linked list.

COALESCING MYFREE()

Improve your myfree() implementation to support the coalescing of two or more small free()'d allocations, so that a subsequent larger mymalloc() can re-use the freed memory.

For example, say I free 4 consecutive allocations of 4 bytes each. A later mymalloc(16) should be able to reuse the free'd memory. To implement this, use the boundary tag method used by the standard malloc.

FASTER MYMALLOC()

When the number of free allocations is large, every call to malloc potentially takes a long time. Change the way you represent free allocations to speed up the process. For this, use an array of free lists for various sizes.

SOME HINTS

- **WARNING:** this will not work on OS X. the brk() system call does not exist on OS X, and is emulated in userspace. This creates a 4 Mb limit on the break size.
- Sometimes, the "best fit" from the free list is still not a very good fit (could be several megabytes off if you're unlucky, especially with a coalescing solution). In that case, simply split your best fit chunk into a great fit, and a newly initialized leftover chunk, that you then add back to the free list.
- When you have a free chunk adjacent to the break, call sbrk with a negative argument to lower the break. Be careful to remove the "adjacent" chunk from your free list!
- Use gdb to debug your segmentation faults!
- If you get a segfault that only happens on the console, and not in gdb, turn on "core dumps" with the command "ulimit -c unlimited". You can then do a retrospective debugging session with gdb thus: "gdb hw7 core", which shows you the status of the program as it crashed.

EXAMPLE OF "GOOD PERFORMANCE", FULL HW7 IMPLEMENTATION OUTPUT

```
./hw7 987987
Simple alloc/free, alloc/free cycle: 0, timediff:237
Coalescing free and splitting mymalloc: 0, timediff:29
After freeing coalesced blocks: -10320000, timediff:7
Random malloc and free sequence: 0, timediff: 698
```

with 1000 16-byte spaced lists:

```
Simple alloc/free, alloc/free cycle: 0, timediff:291
Coalescing free and splitting mymalloc: 0, timediff:67
After freeing coalesced blocks: -10320000, timediff:7
Random malloc and free sequence: 0, timediff: 822
```

with a single list:

```
Simple alloc/free, alloc/free cycle: 0, timediff:239
Coalescing free and splitting mymalloc: 0, timediff:29
After freeing coalesced blocks: -10320000, timediff:7
Random malloc and free sequence: 0, timediff: 2491
```

without coalescing or lowering the break:

```
Simple alloc/free, alloc/free cycle: 100080000, timediff:332
Coalescing free and splitting mymalloc: 9216072, timediff:33
After freeing coalesced blocks: 8192064, timediff:0
Random malloc and free sequence: 497408, timediff: 2535
```

with coalescing, but without splitting:

```
Simple alloc/free, alloc/free cycle: 0, timediff:230
Coalescing free and splitting mymalloc: 8192064, timediff:29
After freeing coalesced blocks: -10320000, timediff:7
Random malloc and free sequence: 0, timediff: 666
```