ICS 53, Spring 2015

Lab 3: A Memory Allocator

The book provides you with a memory allocator which exports 3 functions, mm_init, mm_malloc, and mm_free. The book's allocator is described in Section 9.9.12 and you can download the code from the book's website at http://csapp.cs.cmu.edu/2e/code.html.

You will write a program which allows a user to access the allocator functions (mm_malloc and mm_free), and which allows the user to see the current state of your heap.

You will implement a program which accepts user commands and executes them. Your program should provide a prompt to the user (“>”) and accept the following 8 commands.

1. **allocate** - This function allows the user to allocate a block of memory from your heap. This function should take one argument, the number of bytes which the user wants in the allocated block. This function should call mm_malloc. This function should print out a unique block number which is associated with the block of memory which has just been allocated. The block numbers should increment each time a new block is allocated. So the first allocated block should be block number 1, the second is block number 2, etc. Notice that only the allocated blocks receive block numbers.

Example:

```
> allocate 10

1

> allocate 5

2

>
```

2. **free** - This function allows the user to free a block of memory. This function takes one argument, the block number associated with the previously allocated block of memory. This function must call mm_free.

Example:

```
> allocate 10
1
> free 1
>
```

When a block is freed its block number is no longer valid. The block number should not be reused to number any newly allocated block in the future.

3. **blocklist** - This command prints out information about all of the blocks in your heap. It takes no arguments. The following information should be printed about each block:
   - Size
   - Allocated (yes or no)
   - Start address
   - End address

Addresses should be printed in hexadecimal. The blocks should be printed in the order in which they are found in the heap.

Example:

```
>blocklist
Size Allocated Start            End
8    yes       0x00400000       0x00400007
16   no        0x00400008       0x00400017
4    yes       0x00400018       0x0040001c
```

4. **writeheap** – This function writes characters into a block in the heap. The function takes three arguments: the block number to write to, the character to write into the block, and the number of copies of the character to write into the block. The specified character will be written into n successive locations of the block, where n is the third argument. This function should not overwrite the header of the block which it is writing to, only the payload.

For example, if we wish to write 24 ‘A’ characters into block 3, the user would type the following command:

```
>writeheap 3 A 24
```

5. **printheap** – This prints out the contents of a portion of the heap. This function takes two arguments: the number of the block to be printed, and the number of bytes to print after the start of the block. This function should not print the header of the chosen block, only the payload. If the number of bytes to print is larger than the size of the block, this function should print the bytes anyway, even though they might extend into a neighboring block.

For example, if the user wants to print out the first 10 bytes of block 3, the user would type the following:

```
>printheap 3 10
AAAAAAAAAA
>
```

6. **bestfit** – This function tells the allocator to use the bestfit allocation algorithm from now on. This function takes no arguments.

7. **firstfit** – This function tells the allocator to use the firstfit allocation algorithm from now on. This function takes no arguments.

8. **quit** – This ends your program.

As an additional constraint, change the block alignment to word alignment (4 bytes) rather than double-word alignment.