ICS 53, Winter 2017

Lab 3: A Memory Allocator

You will write a program which maintains a heap which you will organize as an implicit free list. Your program will allow a user to allocate memory, free memory, and see the current state of the heap. Your program will accept user commands and execute them. Your program should provide a prompt to the user (“>”) and accept the following 7 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 payload of the allocated block. 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.

Example:

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

When a block is freed its block number is no longer valid.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:
   I. Size
   II. Allocated (yes or no)
   III. Start address
   IV. 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
```

The information printed should consider any header information. For example, if the user requests a block of payload size 8 and the header is 2 bytes, then the Size of the block should be 10, the Start should point to the first header byte, and the End should point to the last payload byte.

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. The function should not write outside of the payload of the block that it is writing to. If the number of copies passed as an argument to writeheap is larger than the payload size then nothing should be written to the block and the message “Write too big” should be printed to the screen.

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. **printheader** – This prints the header of a block in your heap. This function takes one argument, the number of the block whose header is to be printed. The header should be two bytes long and it should be printed in hexadecimal. For example, the following command might be used to print the header of block 2,

```
>printheader 2
020b
>
```

7. **quit** – This quits your program.

Implementation Details

1. Make the original heap 128 bytes long. The original heap should be configured as a single free block whose size, including header, is equal to 128 bytes. Your code will create the original head using the c library function `malloc()` but this is the only point in your program where you will use the `malloc()` library function.
2. Do not assume double word alignment of the blocks.
3. When a block is requested which is smaller than any existing block in the heap, then your code must perform splitting to create a new block of the appropriate size.
4. Your code should not coalesce freed blocks.

5. Assume that no more than 256 blocks will ever be allocated. Assume that a block is never larger than 128 bytes, including header.
6. The header of each block should be two bytes long. The most significant byte should be the block number. The least significant byte should be the block size and the least significant bit should indicate the allocation status.

Submission Instructions

There will be a dropbox on EEE which you will use to submit your code. You should submit a single C source code file. **The code must compile and execute on the openlab machines**. The name of your C source code file should be “Lab3.c”.