TCSS 333 Spring 2015, HW6

Due: by midnight Friday May 29 (or with 10% late penalty by midnight Saturday May 30)

Create a file called mallok.h The contents of that file should be

```
void create_pool(int size);
void *my_malloc(int size);
void my_free(void *block);
```

You may not modify these prototypes. The work you submit will be tested using a secret grader program written to compile with this header. If you modify these prototypes at all (including the parameter descriptions or the capitalization or spelling of the function names) the grader's program may not compile and your work will be penalized.

Your implementation will be an "abstract object", not an ADT as discussed in Chapter 19. In other words, you will be programming a single global heap, not a heap class. You may use a small number of global variables to implement your heap, but these should be marked "static" to limit access to them. The argument for create_pool indicates how many bytes should be included in the heap. Your implementation of create_pool must use malloc to acquire a single block of exactly this size. This single block is your "private heap". All calls to my_alloc must be satisfied by returning a piece of this private heap if possible. my_alloc should not simply use malloc to allocate the block the caller has asked for. Likewise, my _free should return a block to the private heap, not to the built-in heap by calling free. my_malloc and my_free have the same parameters as malloc and free (although the types have been simplified just a little).

Initially the private heap contains a single free block that has the same size as the call to create_pool.
In responding to my_alloc and my_free calls, this initial block will somtimes be split into smaller pieces and these pieces must later be recombined whenever possible. At any given time, some of the blocks will be in use because malloc has given them to a caller, or they will be free, meaning they are (back) in the private heap. You must build a single linked list to keep track of all the blocks, whether they are free or not. The linked list will be similar to the list you built in HW5, but each node will keep track of a storage block taken from the private heap. For each block, be sure to remember

- where it starts (a memory address)
- how big it is (a number of bytes)
- whether it is free or not

Your linked list of blocks must be kept in order by the starting address of the block. This means that blocks that are next to each other in memory will also be next to each other in your linked list of blocks. To build your linked list of blocks, you will need to create list nodes. You ARE allowed to use malloc in the usual way to create a linked list node. (But as stated above, you are NOT allowed to use malloc to create the blocks themselves.)

my_malloc must search the block list for the FIRST free block that is big enough to satisfy the requested size. If the first free block is more than big enough, then my_malloc should use the first part of the block to satisfy the request and keep the latter part in the private heap. (For example, if my_malloc is looking for a block of 200 bytes and the chosen free block has 1000 bytes starting at address 6000, then the allocated block returned to the caller must be 200 bytes starting at address 6000. A second block of 800 bytes starting at address 6200 is kept in the private heap for possible later use.)

my_free should return a block to the private heap, making that storage once again available. In the process you must recombine the block with any neighboring blocks that are also free.

Calls to my_malloc that cannot be satisfied should return NULL. Calls to my_free that provide an invalid pointer should do nothing. Put your implementation in a file called mallok.c

In a file called mallok_test.c, write code that will test your implementation. Your testing must include at least the following scenarios:

- create a pool of 1000 bytes. Count how times you can successfully request a block of size 10.

- create a pool of 1000 bytes. Make 5 requests for blocks of 200 bytes. Free all 5 blocks. Repeat this request/free pattern several times to make sure you can reuse blocks after they are returned.

- create a pool of 1000 bytes.
  Make 5 requests for blocks of 200 bytes.
  Free the middle block.
  Request a block of 210 bytes (it should fail)
  Request a block of 150 bytes (it should succeed)
  Request a block of 60 bytes (it should fail)
  Request a block of 50 bytes (it should succeed)
  etc.

- create a pool of 1000 bytes.
  Request 5 blocks of 200 bytes.
  Fill the first block with the letter 'A', the second block with 'B', etc.
  Verify that the values stored in each block are still there. (Is the first block full of A's, second block full of B's, etc.)

- create a pool of 1000 bytes.
  Request and return a block of 1000 bytes
  Request and return four blocks of 250 bytes
  Request and return ten blocks of 100 bytes

Your test code will make many calls to my_malloc, often saving the returned pointers and then using those pointers to call my_free. Your test code should print out messages indicating if the different tests are working as expected.

Here is a diagram showing aspects of the implementation. The private heap is shown at the top. Although dotted lines are shown, it is actually one large piece of memory that was obtained from a single call to malloc during execution of create_pool. The nodes of the linked list have additional fields that are not shown.