CSCI 2271 Computer Systems
Assignment 6: Heap Management
Due Friday, March 29

In this assignment, you will build a memory manager for C, replacing the functions *malloc* and *free* with your improved versions *myalloc* and *myfree.* Your code will follow the design described in class. In particular, you should write a file *memorymgr.c* that contains the five functions *initmemory*, *myalloc*, *myfree*, *coalesce*, and *printAllocation*. The declarations of these functions appear in the file *memorymgr.h*, which you can download from the class website. Details of these functions appear below.

initmemory

The function *initmemory* initializes the heap manager. It calls *malloc* to get memory, and initializes that memory to correspond to 2 blocks: a large free block, and a sentinel block (of size 0) at the end of memory. The argument to *initmemory* specifies how many bytes the heap manager must be able to allocate. That is, the size of the user space of the large free block must be at least this big. For example, it should always be possible to write the following program:

```
int main() {
    char *p;
    initmemory(39);
    p = (char *)myalloc(39);
    ...
}
```

In this case, your code will need to call *malloc* (the real one) to allocate 56 bytes: 4 bytes of initial padding, 4 bytes for the header of the 1st block, 44 bytes for the user space of the first block, and 4 bytes for the header of the sentinel block.

This function call allocates all of the memory that your heap manager will use. If the user runs out of space, you are not obligated to try to get more from the OS.

myalloc

The function *myalloc* behaves exactly the same as *malloc*. The code for *myalloc* should scan the implicit list of blocks for a sufficiently large unallocated block. If it cannot find one, it returns NULL. Otherwise, it uses the first one it finds, splitting it into an allocated block and an unallocated block.

myfree

The code for *myfree* behaves the same as *free*. It should simply mark the specified block as unallocated.

coalesce

Your implementations of *myalloc* and *myfree* should not attempt to automatically coalesce free blocks. Instead, I want you to write a function *coalesce*, so that coalescing will occur only when the user requests it. The function *coalesce* coalesces all adjoining free blocks in the block list. This function can be called if *myalloc* returns NULL, in order to create larger blocks.

The reason for this way of doing things is to help with testing (and grading) – It is useful to see when *myalloc* fails, and what happens after coalescing occurs.

printallocation

The function *printallocation* prints the current size and allocation of each block. This function will be useful for debugging purposes, and for demonstrating that your code actually works.

In addition to these five functions, you should also implement the following four functions: *firstBlock*, *lastBlock*, *isAllocated*, and *nextBlock*. These functions also appear in *heapmgr.h*. They will be necessary when you do HW 7. But it makes sense to implement them now, because you can use them in your implementation of HW 6.

The function *firstBlock()* returns a pointer to the first block in your heap space. You use it when you want to search through the blocks. The function *lastBlock()* returns a pointer to the sentinel block at the end of your heap space. You use it to detect when you are done searching. The values of these pointers never change, so your *initmemory* method can save the values in global variables.

The functions *isAllocated(b)* and *nextBlock(b)* provide information about the block located at address b. In particular, *isAllocated(b)* returns 1 if block b is allocated, and 0 otherwise; *nextBlock(b)* returns a pointer to the block following b in the implicit block list.

In your solution, you can implement even more functions to help you out. I encourage you to do so. As you probably know, the best way to write (and debug) a program is to break its tasks into small functions.

Your file *heapmgr.c* will be a library file—that is, it will contain useful functions, but no *main* function. To use the code, you need to compile it with a test file. Such a file is *hw6test.c*, which you can download from Canvas.

Note that *hw6test.c* includes the file *heapmgr.h*. Your *heapmgr.c* file should also include it. Note the you enclose the file in quotes instead of brackets.

To run the program you need to compile the two C files together. You can do this in one step:

```
gcc heapmgr.c hw6test.c –o hw6test
```

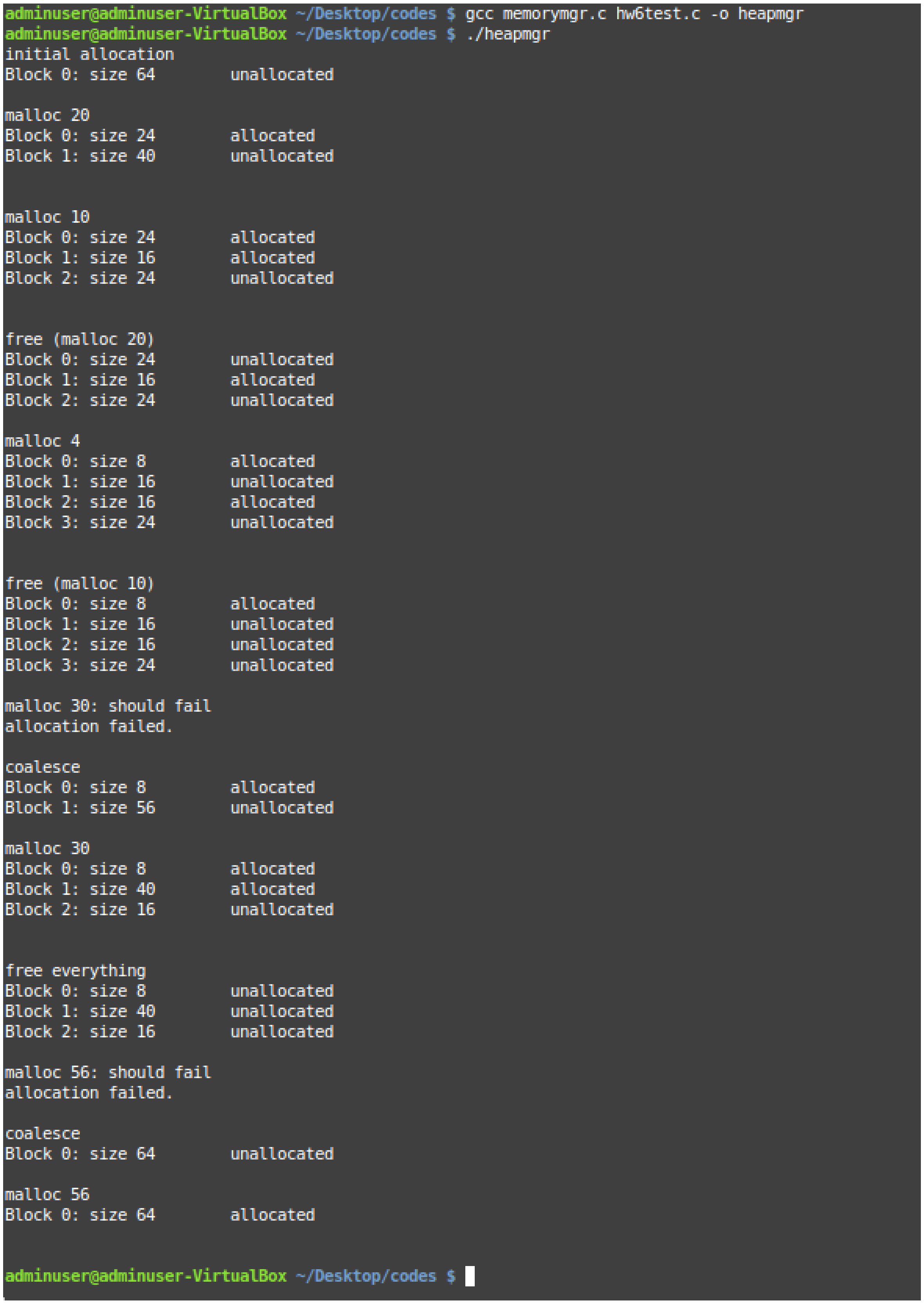
or you can compile each C file separately and then link them together:

```
gcc –c heapmgr.c
gcc –c hw6test.c
gcc heapmgr.o hw6test.o –o heapmgr
```

For your reference, the following page shows the result of running the `heapmgr` program in the Linux environment.

When you are finished, submit your file *heapmgr.c* to Canvas.

WARNING: This is a significantly more difficult assignment than you have seen so far. Start early! Learn to use the debugger!



```
adminuser@adminuser-VirtualBox ~/Desktop/codes $ gcc memorymgr.c hw6test.c -o heapmgr
adminuser@adminuser-VirtualBox ~/Desktop/codes $ ./heapmgr
initial allocation
Block 0: size 64        unallocated

malloc 20
Block 0: size 24        allocated
Block 1: size 40        unallocated


malloc 10
Block 0: size 24        allocated
Block 1: size 16        allocated
Block 2: size 24        unallocated


free (malloc 20)
Block 0: size 24        unallocated
Block 1: size 16        allocated
Block 2: size 24        unallocated

malloc 4
Block 0: size 8         allocated
Block 1: size 16        unallocated
Block 2: size 16        allocated
Block 3: size 24        unallocated


free (malloc 10)
Block 0: size 8         allocated
Block 1: size 16        unallocated
Block 2: size 16        unallocated
Block 3: size 24        unallocated

malloc 30: should fail
allocation failed.

coalesce
Block 0: size 8         allocated
Block 1: size 56        unallocated

malloc 30
Block 0: size 8         allocated
Block 1: size 40        allocated
Block 2: size 16        unallocated


free everything
Block 0: size 8         unallocated
Block 1: size 40        unallocated
Block 2: size 16        unallocated

malloc 56: should fail
allocation failed.

coalesce
Block 0: size 64        unallocated

malloc 56
Block 0: size 64        allocated


adminuser@adminuser-VirtualBox ~/Desktop/codes $ 
```