CS 3410 Malloc Documentation

Lylla Younes (ley23) and Nazmus Sarwar (ns592)

Approach.

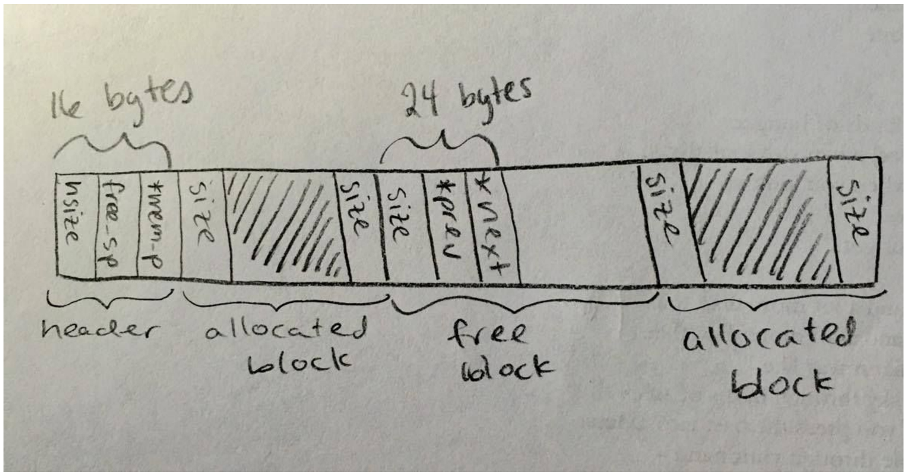
We decided to use an explicit free-list with limited metadata and coalesced blocks. The heap header is simple. We pad the pointer to the heap until it is 8-byte aligned (if it is already aligned properly we do not add any padding). We created a struct called “heap_header” which stores three important fields:

1. h_size: stores the size of the heap
2. free_space: stores the amount of free bytes in the heap
3. mem_ptr: pointer to the root node of the explicit free list

In terms of metadata for allocated blocks, we store the size of the entire block at the beginning of the block (this includes all padding and metadata) and we store the size of the payload at the end of the block. We created a struct called “block” which is stored before every free block and which contains three fields:

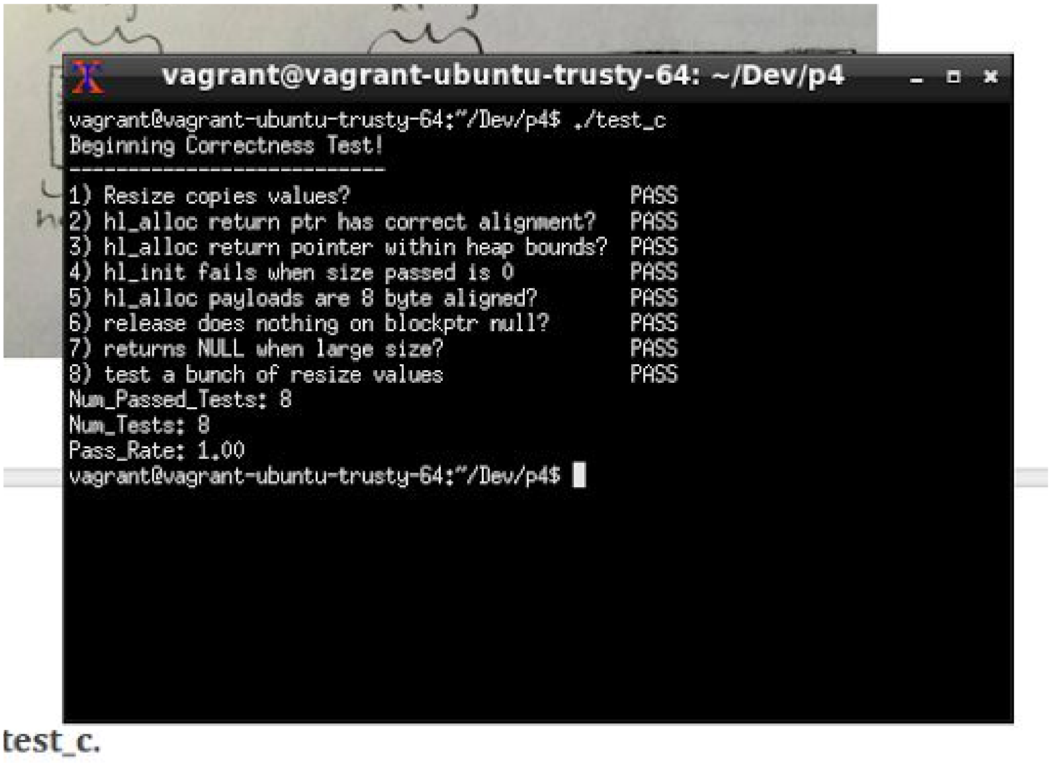
1. size: the size of the free block
2. prev: pointer to the previous block in the free list
3. next: pointer to the next block in the free list

Below we include an example drawing of what our heap might look like:



test_c.

Show a printout of the *output* of test_c so that the staff can see how many correctness tests you created and how your library performs on your own tests. Highlight anything unusual or particularly interesting.



```
vagrant@vagrant-ubuntu-trusty-64:~/Dev/p4$ ./test_c
Beginning Correctness Test!
---------------------------
1) Resize copies values?                        PASS
2) hl_alloc return ptr has correct alignment?   PASS
3) hl_alloc return pointer within heap bounds?  PASS
4) hl_init fails when size passed is 0          PASS
5) hl_alloc payloads are 8 byte aligned?        PASS
6) release does nothing on blockptr null?       PASS
7) returns NULL when large size?                PASS
8) test a bunch of resize values                PASS
Num_Passed_Tests: 8
Num_Tests: 8
Pass_Rate: 1.00
vagrant@vagrant-ubuntu-trusty-64:~/Dev/p4$ 
```

sizetask.c

For this task, we have a lot of calls to alloc, resize and free.
We want to see how much our heap we are actually able to use.
So calling resize and release will allow us to see how much
internal fragmentation there is.

We fixed a seed to a random number generator (so that our version of
pass_sizetask will run the same random numbers on each students' heaplib.c).
The randomness component demonstrates how well our heap can perform
utilization-wise in circumstances where calls to alloc are not conveniently
placed right after one another.