CSSE 332 -- Operating Systems

Heap Manager I

Name: _____

If you have not read the lab instructions and watched the overview videos yet, please do so before answering any of the below questions; they would not make much sense otherwise.

Question 1. (5 points) In this lab, all of the size arguments passed to `rhmalloc` must be transformed into the nearest _____.

Question 2. In the following questions, assume that we use the `sbrk` system call to ask our operating system for an initial slab of memory of size **256 bytes**.

Furthermore, assume that our design uses a metadata structure that contains three elements:

1. `used`: A Boolean flag that indicates if the slab is in use.
2. `size`: The size of **usable** area of the slab.
3. `next`: A pointer to the next slab of memory, if any (can be `null` as well).

After running some experiments on our architecture, we find that the size of this metadata structure is **8 bytes**.

(a) (5 points) Describe, using visuals if you like, the initial content of your `freelist` before any calls to `rhmalloc` are made.

(b) (5 points) Draw you `freelist` after a user makes a call to `rhmalloc(16)`.

(c) (5 points) This part continues right after part b. Draw you `freelist` after a user makes a call to `rhmalloc(28)`. In other words, our user has made two calls to `rhmalloc`, one for 16 bytes and another for 28 bytes.

(d) (5 points) Assume now (after part c) that the user calls `rhmalloc(185)`, would the function call be successful? Explain why or why not.

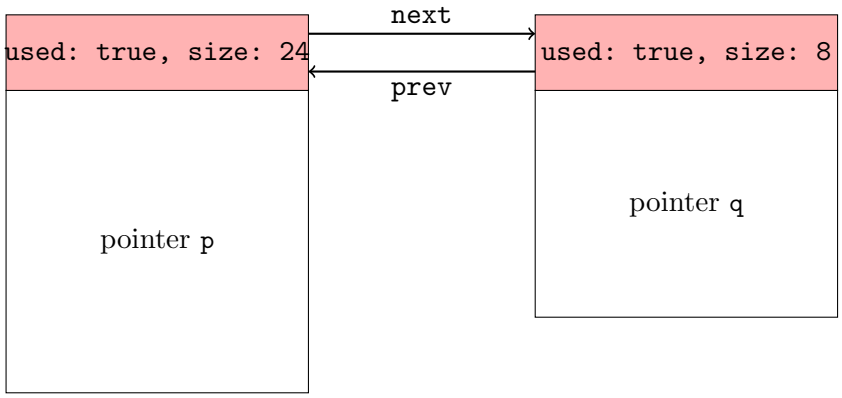
(e) (5 points) The user now is done with the first pointer they requested (the one from part b) and they call `rhfree` to free it. Draw the status of your `freelist` after `rhfree` executes.

(f) (5 points) Assume that the user now also frees the pointer allocated in part c, i.e., the user has free'd all requested memory and everything should be free again.

If not coalescing was implemented in `rhfree`, would a new call to `rhmalloc(185)` be successful? Why or why not? You might find it useful to draw the state of your `freelist` at this stage.

(g) (5 points) In the lecture, we discussed a condition under which a free block could not be split in two, and thus we ended up wasting some bytes over the amount that the user has requested. In the box below, write down a generic condition under which a free block of memory *can be split in two*.

Question 3. Consider the following `freelist` snapshot:



Assume now that the user issues the following calls:

```
rhfree(q);
rhfree(p);
```

(a) (5 points) Describe how the blocks will be free'd and then coalesced. Draw the status of the `freelist` after each call. Make sure to label the content of the metadata structures in each block available.

(b) (5 points) Write a **pseudocde** that captures the steps necessary to merge two blocks in the `freelist`. Make sure to account for the following:

1. Adjust the `next` pointer in each block as well as their neighbors.
2. Adjust the `prev` pointer in each block as well as their neighbors.
3. Adjust the `size` of the created blocks.