COMPUTER SYSTEMS
PROJECT 1
Allocator, Mark II: Reclaiming Free Space

1 Enhancing our allocator

1.1 Overview

Begin with the *pointer-bumping allocator* of Project 0. Here, allocation was performed in a *first-fit, address-ordered* manner with *no reclamation of free space*. It was simple, but it worked.

For this assignment, we want to alter that allocator. First, it **must reclaim free space**. Second, it **must allocate from the collection of reclaimed blocks preferentially**. That is, if the allocator can fulfill a request by using a block that was previously made `free`, then it must do so. Pointer bumping allocation should only be employed when no reclaimed blocks can satisfy the request.

1.2 A suggested design

To make these changes, `free()` must track all blocks passed to it, and `malloc()` must find and use those blocks. One approach that would be straightforward is:

- Keep a **free list of reclaimed blocks**. `free()` inserts blocks into this list; `malloc()` traverses the list and potentially removes a block from it.
- Implement this list as a **linked list**, where the nodes of the list are stored *within* the free blocks themselves.
- Have `malloc()` employ a *first-fit* approach to satisfying requests from the free list.
- Do **not** have `malloc()` split free blocks. That is, if a request of size n can be satistfied with a free block of size m, where $n < m$, then `malloc()` should return the enter m-byte block. It should **not** divide the block into an n-byte block for the request, and then an $(m - n)$-byte block that remains in the free list.
- Do not attempt to coalesce free blocks.

1.3 How to test it

Remember that you can compile `pb-alloc.c` such that it is a standalone program that you can not only run, but run inside of *GDB*.[1] Specifically, compile it like so:

```
$ gcc --std=gnu99 -ggdb -o pb-alloc pb-alloc.c
```

Given this ability, you should write `main()` such that it calls `malloc()` and `free()` with a fixed sequence of requests that should exercise the various parts of your allocator. I leave it to you to determine what sequences to use. I recommend only that you **progress incrementally**. The first attempt should allocate, say, 16 bytees. That's it. Then it should allocate and then free that 16 bytes. Next, allocate, free, and allocate 16 bytes. The second allocation should yield the same block that the first allocation did, reusing that block.

Finally, **use the Google**. How do you know if the same block was used? You could print the pointers returned by `malloc()` and see if they're the same! How do you print pointers in C? I repeat: use the Google.

2 How to submit your work

Submit your new, enhanced `pb-alloc.c` file. You may use either of the following two methods to use the CS submission system:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `cssubmit` command at the shell prompt on `remus/romulus`.

This assignment is due on Sunday, Oct-15, 11:59 pm.

[1] Some have been resistent to get familiar with GDB. Get over it. Go dig up documentation and tutorials, and learn to use it.