Computer Systems
Project 5
A (simulated) garbage collector

1 Writing a mark-sweep garbage collector

C is not normally amenable to garbage collection. As a language, it does not maintain information about how objects are laid out in the heap. Specifically, it provides no information about where pointers are stored within arrays and objects, so there is no way to follow the pointers in performing the reachability traversal (the *depth first search* of the heap).

For this assignment, we will fake it. We will have the ability to create structures that store information about the objects we allocate on the garbage collected heap (henceforth, the *GC heap*). A tester program can then create these *layout* information objects, and then allocate them on the GC heap with that layout information attached.

From there, the tested can manually provide the GC the root set, and then call on the collector to do its job. This hand-holding of the collector is not normal—it's a simulation of what a *programming language runtime environment* would do automatically in a language designed for garbage collection.

Once the collector is invoked, though, the collection is real. It must traverse the GC heap, marking each live block it encounters. It then must sweep the allocated objects, freeing any that are not marked. That activity is the focus of your work on this project.

2 Getting started

2.1 Creating the repository

1. **Login to the server** via `ssh`.
2. **Login to GitLab** in your browser.
3. **Start a new project:** Set the *Project name* to be `sysproj-5`.
4. **Clone the repository onto the course server:**

```
$ git config --global user.name "Your Name"
$ git config --global user.email "yourusername@amherst.edu"
$ git clone git@gitlab.amherst.edu:yourusername/sysproj-5.git
$ cd sysproj-5
```

5. **Download the source code:**

```
$ wget -nv -i https://bit.ly/cosc-171-20F-p5
$ ls -l
```

6. **Add/commit/push the source code to the repository:**

```
$ git add *
$ git commit -m "Starting code."
$ git push
```

2.2 Compiling and running

Compiling and running this code is simpler than for Project-3 and -4. Since the GC heap is simulated, all of the code is compiled into a regular executable program, and you just run it. So:

```
$ make clean gctest
```

Then, you run `gctest` itself, providing the simple information of a number of objects for the tester to create before calling the collector:

```
$ ./gctest 10
```

3 Your assignment

All of your work will be in `bf-gc.c`, where you will write your garbage collector code on top of your *best fit* allocator. You may also modify and enhance `gctest.c`, but more on that below.

3.1 Part I: Port your core allocator functions

From your `bf-alloc.c` code from Project-4, copy-and-paste the your `malloc()` body into the `gc_malloc()` function in `bf-gc.c`. Likewise, copy-and-paste your `free()` from `bf-alloc.c` into `gc_free()` in `bf-gc.c`.

Be a little careful here. You likely defined things like static variables and `#define` macros in `bf-alloc.c`. You will need to copy those over, too. Check your code for such small adjustments, but overall, these functions should largely drop into place.

3.2 Part II: Write the collector functions

The `gc()` function simply calls on two functions: `mark()` and `sweep()`. You need to write these two functions.

The `mark()` function should be an adaptation of the psuedocode from our lectures on traversing the heap. That pseudocode was for a copying collector; here, there's no copying or pointer updating, just the marking of blocks that we reach. Therefore, the depth first search stack is of *pointers*, and not of *handles*.

Notice that the *root set* is a linked-list implementation of a stack, and that there are pre-made internal functions (`rs_push()` and `rs_pop()`) to use it. You should use this stack to keep track of the pointers that need to be followed. When a collection starts, you should assume that this stack already contains the root set of pointers.

Also notice that each block's header now has a `layout` field. This pointer leads to a `gc_layout_s` structure that provides information about how many pointers are in the block, and where (relative to the base address of the block) to find them. You can thus find these pointers and *push* each onto the search stack.

The `sweep()` function requires a linear traversal of the *allocated list* that you created for Project-4. For each block on that list, either it is marked, in which case you can unmark it (for the next collection), or it is **not** marked, in which case it is dead and must be freed with `gc_free()`.

3.3 Totally optional challenge

If you get the above to work and want to be adventurous, create a `sf-gc.c` source code file based on your segregated fits allocator from Project-4. Adapt the changes for garbage collection from the *best-fit* allocator to this allocator.

4 How to submit your work

First, be sure that the most recent versions of your work are up-to-date on the GitLab server by performing an *add/commit/push* with `git`. Then, go to GitLab with your browser, and add me (*sfkaplan*) as a *Developer* to your repository.

This assignment is due on Sunday, Oct-25, 11:59 pm.