home

syllabus

schedule

labs

- week 1 lab
- week 2 lab
- week 4 lab
- week 5 lab
- week 6 lab
- week 7 lab
- week 8 lab
- week 10 lab
- week 11 lab
- week 13 lab

homeworks

- homework 1
- homework 2
- homework 3
- **homework 4**
- homework 5
- homework 6

lectures

- Syllabus, tricks of the trade intro

discussion

gradescope

Homework 4: A garbage collector for C

Assignment checkout

Background

Dynamic memory (i.e. memory accessed via the `malloc` and `free` family of commands) is essential to many programs in the freedom, control, and interpretability it allows. We've talked about 2 different types of management for dynamic memory: implicit allocators and explicit allocators. This assignment focuses on the implicit allocator, perhaps better known as the *garbage collector*. Intuitively, we want this tool to find the blocks (or chunks as we refer to them) that are not being used and return them to the process as free chunks.

For further reference, please see section 9.10 (and specifically 9.10.2) of the book.

Memory layout review

As a quick review, we can visualize dynamic memory (aka the heap) as a list of chunks. From class, we know that each allocated or free block is started with header (shown as 'h' in the figure below), contains a payload ('x'), and has some form of padding at the end.

```
----------------------------------------------------------------------
| h  | x  | x  | h  |    | h  | x  | h  |    |    |    |    |    ...
----------------------------------------------------------------------
^              ^         ^         ^
|              |         |         |
a              b         c         the rest of the heap...
```

In the figure, 'a' represents an allocated section of the heap with a payload + padding taking up 2 words worth of space. Similarly, 'b' is a free node of size 1 word.

While you will not need to delve into the details of this implementation or structure to complete homework 4, it will be helpful to keep this image in mind while you implement your garbage collector.

Chunk structure

```
typedef struct chunk {
         size_t size;
         struct chunk* next;
         struct chunk* prev;
} chunk;
```

The programming part

In this homework, we build a basic, conservative garbage collector for C programs. Starting from the set of root pointers present in the stack and global variables, we traverse the "object graph" (in the case of malloc, a chunk graph) to find all chunks reachable from the root. These are marked using the third lowest order bit in the size field of each chunk. We have implemented several helpful functions for you ranging from checking for marked chunks to navigation of the heap. Please read the skeleton code to get a feel for what is available and what you should be doing. Ultimately, you will need to implement the sweeping functionality of the garbage collector, the pointer-confirmation functionality, and a portion of the marking functionality.

As of the release of this assignment, we have not covered the concepts related to garbage collection in class; to start on this homework, take a look at section 9.10, and 9.10.2 specifically. The week 10 lab will help with getting started on the assignment as well.

The skeleton code in the public git repository provides supporting code for finding the limits of the global variable area in memory, as well as for the stack and heap areas. It also demonstrates very basic marking and sweeping. Your task is to complete the garbage collector, so that it frees all garbage, and leaves every other chunk intact. To compile the template, type `make`, and run it with `./hw4`.

Note that all of the tests have descriptions in `main.c`! Use these to give you some hints on what kind of functionality your garbage collector should have. Also, take a look at `memlib.c` and `memlib.h` to get a feel for what kinds of memory operations you have available to you.

Requirements

For full points, your program should pass all tests in `main.c` - that is, it should free all inaccessible memory. At present, these are not coded as actual tests, they simply output the heap size and number of free and in-use chunks. Nevertheless, your program should produce the correct values. Attempt to run the skeleton code to get a feel for what this output should look like. Remember: your goal is to implement a garbage collector, after it is run each time, should there be more or less free chunks?

Runtime performance is not a major goal for this homework. However if your program is taking more than 2 seconds, you almost certainly did something wrong.

You should only make changes to `hw4.c` in the areas marked TODO.

Finally, *make sure you do not output anything extra in your implementations*. This will break the autograder.

Grading

The completed assignment should be turned in via Gradescope as with other homework assignments. Please test your code before turning it in and do not use the autograder as a debugging tool.

Due Date

The assignment is due Mon, 09 Nov 2020 22:59:00 -0600. See the syllabus for the late turn-in policy.