Goal

Study the influence of different heap management options.

Problem Statement

Sponge Bob, Patrick, and Squidward each have a favorite technique for managing the heap (they all agree that Mr. Crabs stacks are to die for). Sponge Bob asserts that all memory blocks should be the same size, so he wrote his own constant-size allocator (well with your help that is), Patrick likes the one that is built into Sea, and Squidward prefers for someone else to take out the garbage (we will let Java to this). Java uses an implicit heap manager that attempts to determine what, if any, of the memory that was previously allocated is no longer accessible, which is referred to as “garbage”.

Analysis (What is the client’s problem)

Software Engineers first construct an analysis (what is it) and then a design (how will it be done).

The first thing to understand is the kind of heap being considered (this is more an issue for those of you who have survived CS 312, which looks at Binary Heaps (simply called heaps in that class). In CS 366 we consider Heap Memory (also commonly referred to simply as a “heap”). While a binary heap includes items (e.g., integers) in each node and has a prescribed order, heap memory does not. Consequently it makes no sense to talk about “sorting the heap.” For more detail, take another look at Section 9.9.

A second analysis issue is what is meant by a constant-size allocator. Simply put, such an allocator allocates the same amount of memory each time it is called.

For this assignment we will investigate which algorithm works the most efficiently. The program has no “user interface” as such. Instead for this experiment, your test driver will allocate and deallocate block of memory randomly using the following pseudo code:

```java
create an array of BLOCK_COUNT entries
for ITERATIONS iterations
  pick a random block
  if that block is allocated, free it
  otherwise allocate BLOCK_SIZE bytes and assign it to the block
```

The number of iterations (ITERATIONS), the size of the blocks (BLOCK_SIZE), and the number of initially available blocks (i.e., BLOCK_COUNT, the initial size of the heap) should be specified using compile-time constants.

A final good analysis question here is “What measures should be used to compare the three heap management techniques?” For this assignment, the answer is the CPU time and wall-clock time taken to run the test driver (check out /usr/bin/time and the target test found in the Makefile).

[Commit any further analysis question as part of README.md before moving on to the design.]
Design (How will this problem be solved)
In this case the big how question is “how will you and Sponge Bob maintain your custom heap?” Possible answer to this question involve lists, stacks, or perhaps trees.

[Commit your design (as part of README.md) before moving on to the plateau schedule.]

Plateau Schedule

Before you start coding take the time to write out a build plan.

There are actually two of these for this assignment. The first is for the experiential study and the second is for the construction of your constant-size allocator. While you can create your own schedule, I would get the experiment working using Patrick’s allocator before coding the constant-size allocator.

[Commit your plateau schedule (as part of README.md) before moving on to the code.]

What to hand in (deadline [points]) (Please no .docx files.)

1. 6am 3/19 [10]. I will pull your analysis and then push comments.
2. 6am 3/21 [10]. I will pull your design and plateau schedule and then push comments.
3. Class time 3/29 [5]. A well-formatted 2-up printout of your C and Java source code. You must use a2ps after removing all the tabs from your code. Indent code 2 or 4 spaces at most.
4. Class time 3/29 [75]. A GitHub repo that includes (you must use these names as the grading script will assume their use!)
   - README.md with the sections Analysis, Design, Plateau Schedule, and Empirical Investigation (i.e., your take on the generated data).
   - main.c and JavaWay.java, and
   - Makefile (where make all and make test will build and test your code).

Assignment Notes
• You must use #define macros for the two heap functions get_mem(⋯) and free_mem(⋯). The macro get_mem(⋯) returns a block of memory from the heap, while the macro free_mem(⋯) returns a block to the heap where it is available to be returned by a subsequent get_mem(⋯) request.
• You must use #ifdef to enable the compile-time selection between Sponge Bob’s way and Patrick’s way as required by the Makefile. The #ifdef should guard the definitions of get_mem(⋯) and free_mem(⋯) for the two non-Java approaches.
• By special permission during the season of lent, your code can include static (file scope) variables.
• Target a run time for Patrick’s version of about 5 to 10 seconds (by setting ITERATIONS).
• Better answers will experiment with the impact of BLOCK_COUNT and BLOCK_SIZE as well as any initialization of Sponge Bob’s data structures (consider using command-line arguments).
• I expect to pull your code, run make, and then run my test script.
• You must use memwatch with all C (and Sea) code. Better answers will report “unfreed bytes 0”. However, once your program works, comment out the use of memwatch before setting up and running the timing experiments.
• The output of git log will again factor into your grade.
• Include in README.md, under the heading Empirical Investigation, the maximum memory usage for each of the three using a common configuration. Repeat this comparison for a few other configurations. running in a separate window. Format this information nicely!!
• Reminder! run the final time trials without memwatch! (e.g., edit the `Makefile; make clean; make all; make test`.)

• Here is the git classroom invitation
  https://classroom.github.com/a/ud6yWScK

• Don’t forget to free the cache!

• Want to have more fun? Plankton’s allocator includes “```#define free_mem(x)```” because freeing memory is a waste of time. Is he right?

• Initial grading rubric includes, but is not limited to
  10 analysis
  10 design
  5 printout
  75 repo
  5 log
  5 max mem recorded
  5 #ifdef / #define functions
  20 (exp) explanations of BLOCK_COUNT and BLOCK_SIZE impact
  40 src