Garbage collection

Parts I and II due Tuesday, March 1, at 11:59 PM.
Part III due Thursday, March 3rd, at 5:00PM, outside of Maxwell Dworkin 133.

The purposes of this homework are

- To review the implementation of mark−and−sweep garbage collection, which you may have seen in CS 51.
- To understand how to bound mark−and−sweep collection times by interleaving the sweep phase with the allocator.
- To understand how to implement a simple copying collector.
- To study and understand the effects of heap size on performance in both mark−and−sweep and copying collectors.

Although the mark−and−sweep collector is the first one discussed in the book (because it is easier to understand), you may well find your *implementation* is easier if you begin with the copying collector.

Theoretical preliminaries: The steady−state assumption

In Part III, we ask you to analyze the behavior of your garbage collectors. Accurately modeling and analyzing the behavior of real collectors on real programs is a problem that has yet to be solved. But you can get a useful approximate model by using the *steady−state assumption*. This assumption says

> After an initial *startup phase*, a garbage−collected heap reaches a *steady state*.

To understand this assumption, recall that the mutator (a program) allocates at will, and after any allocation request, the system may decide to collect garbage. Therefore, a program's memory−related behavior can be divided into a sequence of *allocation cycles*, each of which consists of a sequence of allocation requests followed by a garbage collection. (When generational collection enters the picture, we have cycles within cycles.)

At startup time, there is no live data, so the heap is empty. The steady−state assumption says that, after some number of cycles in which the heap grows, eventually it reaches a steady state: that is, at the end of each collection cycle, the heap looks pretty much the same as it looked at the end of the previous cycle−−−provided it has reached the steady state.

Practical preliminaries: Setup & Interpreters

Inside your local `cs152` directory, create a directory called `gc`. Go to that directory. Copy the code for both collectors as follows:

```
cp -r ~cs152/software/bare/uscheme-copy .
cp -r ~cs152/software/bare/uscheme-ms   .
```

Some of the code will contain lines marked with the mysterious comment `/* OMIT */`. This comment marks secret code that doesn't show up in the book but is useful for homework. You are welcome to use it.

Part I: Copying garbage collection

From Ramsey and Kamin, do problems 12–14 and problem 15, parts (a) and (c). [25 points]

For problem 14, please print your statistics in this form:

```
[GC stats: heap size 192 live data 91 ratio 2.11]
[Mem stats: allocated 389 heap size 408 ratio 0.95]
[Total GC work: 7 collections copied 642 objects; 1.65 copies/allocation]
```

Please use exactly two digits after the decimal point when printing ratios; if there is no live data, print the ratio as `infinite`.

For problem 15, be sure to ***explain your choice of gamma.***

Our solution to part I requires about 150 lines of C code, of which only a small amount is devoted to debugging.

Part II: Mark–and–sweep garbage collection

From Ramsey and Kamin, do problems 1–4 and problem 5, parts (a), (b), and (d). [30 points]

You may find it helpful to do problem 4 before attempting the implementation. Your solution to problem 4 can be written in ASCII.

For problem 2, please print your statistics in this form:

```
[GC stats: heap size 168 live data 105 ratio 1.60]
[Mem stats: allocated 389 heap size 192 ratio 2.03]
[Total GC work: 6 collections marked 469 objects; 1.21 marks/allocation]
```

As above, please use exactly two digits after the decimal point when printing ratios; if there is no live data, print the ratio as `infinite`.

Our solution to part II requires about 150 lines of C code, of which about 60 lines are devoted to debugging.

Part III: Performance analysis

From Ramsey and Kamin, do problem 5, part (c), problems 6 and 7, problem 15, part (b), and problems 17 and 18. [45 points]

Present your solutions to these problems in a written report. Please submit your report on paper by 5:00PM Thursday, March 3rd.

- Your report should be organized in a logical way that leads the reader to understand the problem, your experimental method, and the results of your measurements.
- The report should explain how well your formulas predict your measurements, the reasons for any discrepancies, and any steps you may have taken to reduce discrepancies.
- The report should give any conclusions you are able to draw about work per allocation and comparisons of mark–and–sweep with copying.

- Avoid the common mistakes listed in the handout on experimental methods. If the nature of a mistake is not clear, consult a member of the course staff.

Note: you may not test solely with insertion sort and merge sort; *you must choose at least one additional test program*. You should look for a program that exercises the memory system in interesting ways.

Some of the problems call for formulas that express various costs as functions of gamma. If you find it easier to do the analysis by thinking about other quantities, such as the number of cells allocated or the fraction of the heap occupied, fine, but before you turn in your work, *all* other quantities must be rewritten in terms of gamma, so that gamma is the only property of the heap that appears in your final answer.

Hint: for problems 7 and 18, try to adjust your experiments to make your results as consistent as possible with your formulas. Your report should *say what adjustments you made and why*. Consider the steady−state assumption.

We have prepared a handout that explains how to present the experimental data you will need to gather for problems 5(c), 7, 15(b), and 18. You may also find it useful to use Alex Kulesza's perl scripts to help you gather data and convert to jgraph form.

Supporting code and notes

To help with your measurements, we are providing code that will spit out test cases using merge sort and insertion sort. If you call `~cs152/bin/mergetest 18` it will print out a definition of merge sort and a call to sort an 18−element array, so you can try

```
~cs152/bin/mergetest 56 | ./uscheme
```

for example, to try out your own interpreter. A similar script called `inserttest` lets you try the exact same test but using an insertion sort. You will probably want to sort arrays of many different sizes as you gather your measurements. *You should also look for other programs to measure.*

We reserve the right to use different values of `GROWTH_UNIT` to exercise your collector, so **you should be sure that your program works with any positive value of `GROWTH_UNIT`**.

It is quite difficult to test a garbage collector, so ***the bulk of your grade will be based on your ability to convince us that you have implemented a correct collector.*** Don't forget to explain what you have done, and don't leave your explanation for the last minute!

Advice

Garbage collectors are hard to debug. The best advice we can give you is *think first, code later*. The temptation to dive into the code after noticing a bug can be very strong. Resist it. You have a puzzle in front of you: the interpreter produces the wrong answer when you type some particular expression. Experiment with similar expressions. Try to find the conditions under which the bug is exhibited. If the problem happens when a particular expression is given as an argument to a user−defined function, does the problem occur when given to a primitive function? If the problem happens using the `val` statement, does it happen if you use `set` instead? And so on. Before you go into the program, you should have gathered enough evidence from probing the interpreter to incriminate some specific section of code.

Extra credit

For *massive extra credit*, you may attack problem 11, 20, 22, 23, or 24 from Ramsey and Kamin. If you solve problem 11, major bonus points if you manage *all* objects using your conservative collector, not just `Values`.

For *modest extra credit*, you may attack any other problem from Chapter 4 of Ramsey and Kamin. It is the instructor's opinion that problems 16 and 19 are probably the easiest of the remaining problems.

How to prepare your code and what to submit

To set up your directory, do the following:

```
cd ~/cs152
mkdir gc
cd gc
cp -r ~cs152/software/bare/uscheme-copy .
cp -r ~cs152/software/bare/uscheme-ms   .
```

This software is part of the textbook software distribution.

Whether you solve all the problems or only some, we expect you to submit at most two collectors. This is what we are expecting for Parts I and II:

- Submit a `README` file that
  - Tells us what parts of the assignment you have completed
  - Says with whom you collaborated
  - Says about how many hours you needed
  - Explains your solutions to Parts I and II
  - Includes your solution to Exercise 4
  - Says what you measured to arrive at a default value of gamma
  - Says what default values of gamma you decided on
  - Tells us anything else we should know about the assignment or your work
- If you wish to try your luck at breaking our code, or anyone else's, you may submit up to three tests in a file `tests`. We've prepared an example `tests` file you can use to see what the right format is.
- In subdirectory `uscheme-copy`, you should include your copying collector. *It should not be necessary to change any file except `copy.c`.*
- In subdirectory `uscheme-ms`, you should include your mark–and–sweep collector. *It should not be necessary to change any file except `ms.c`.*

  Submit using `submit-gc`.

Submit your solutions to Part III on paper. Please include a note indicating how many hours you spent on Part III.