22c:111 (CS:3820) Programming Language Concepts
Fall 2014
Homework Assignment 3
Sample Solution
Due: Wednesday, Dec 10 at 11.59pm

1 Stop-and-Copy Garbage Collector (50 points)

For this exercise we are implementing a copying garbage collector as seen in class.

A copying garbage collector partitions the heap in two halves, called the *From*-space and the *To*-space. All allocations happen in the *From*-space, the *To*-space is not used. Garbage collection is performed by copying all live objects on the *From*-space to the *To*-space, then *From*-space and *To*-space swap their roles.

Cheney's Algorithm We follow Cheney's algorithm, which works as follows. See, `http://www.cs.umd.edu/class/fall2002/cmsc631/cheney/cheney.html` for an interactive animation.

Copy the objects referenced in the root set to the *To*-space. At the former location of an object in the *From*-space leave a forwarding pointer with the new address of the object in the *To*-space and replace the pointer in the root set with the new address of the object in the *To*-space.

Scan each object in the new *To*-space, and follow each reference into the *From*-space. If you find a forwarding pointer, replace the reference in the object with the forwarding pointer. Otherwise copy the referenced object to the *To*-space, leave forwarding pointer at the address, and replace the reference with the new address of the object.

Stop when all objects on in the *To*-space have been scanned. The algorithm ensures that all objects reachable from the root set are copied to the *To*-space, all pointers refer only to the *To*-space, and the *To*-space is compact, that is, contains no holes between allocated objects.

Implementation We model the memory as an array. See `memory.ml` for the implementation. Each cell in the memory contains a value of type `cell`, which marks the cell as free, is a forwarding pointer to the *To*-space, or an object. Objects can be several words long, and contain arbitrary number of references. For simplicity, the first word allocated for an object is a value `Object (uid, size, refs)`, where `uid` is a unique identifier for the object, `size` is its size in words, and `refs` is a list of references. An object may occupy several words in the memory, that is, an `Object (1, 3, [])` is of size three and thus followed by two `ObjData 2`, and `ObjData 3` values.

The memory itself is in the array `ram`, you can assume that its size is even. The first half of `mem` is the *From*-space, the second half is the *To*-space.

Type your code into the template files `copyingGC.ml` that is provided along with this assignment. You must implement all three functions `copy_gc`, `scan_tospace` and `copy_obj`.

The first function `copy_obj` copies the object at `addr` to `free`, unless it is a forwarding pointer. In the first case, it returns an updated `free` pointer, and the new address of the object. It must copy the whole object as of its size. If the object is a forwarding pointer, it returns the `free` pointer unchanged and value of the forwarding pointer.

The function `scan_tospace` is recursive, ideally tail-recursive, and takes as arguments the next free memory location in the *To*-space (`free`), and the location of the first unscanned object in the *To*-space (`unscanned`). It checks all references of the object, updates them with references in the *To*-space by either copying the referenced object, or reading the forwarding pointer. Use the `copy_obj` function. Terminate and return the `free` pointer, if the `unscanned` pointer is equal to it.

The top-level function `copy_gc` is called with a list of references in `root_set`. Copy the objects referenced from the root set to the *To*-space using `copy_obj`, initialize `free` and `unscanned`, and call `scan_tospace`. Return the root set with the references updated to the *To*-space in the order they were input.

You will find two additional files with this assignment, `copyingGCChecker.ml` and `Makefile`. Do not modify or submit these, they are meant to give you a preview on how your submission is assessed. We will use different test cases for the actual grading. Open a terminal window in Linux or Mac OS X, or use the Cygwin terminal in Windows, and type `make` to compile your sources and run some test cases.

Before you submit make sure that your code compiles and run `make test` *at least once. You will get zero points otherwise.* (50 points)