

# CS245 Lab 5: Reference Counting

*Due Monday of the 11<sup>th</sup> week of classes (16 Nov 2009); handed out on the 9<sup>th</sup> Friday.*

Most of you will need to spend a significant amount of time just figuring out the details of C++ constructor and destructor functions in this lab. This has the potential to be tedious and frustrating, so you are invited to sign up to work in pairs for Steps 4-6, as long as you agree to make it an exercise in “pair programming”. In other words, you will both be present for all the work, will frequently trade roles, and will ensure that each member of the pair spends about half of the total computer time at the keyboard. All teams for pair programming must send email by Wednesday of the 10<sup>th</sup> week of classes. You are also welcome to work on your own if you prefer or are having trouble coordinating with your classmates.

Obtain the starter files for the `List-Memory-Management` project. The starter files for this project define the linked list class with “do lots of copies so that each link object has only one pointer to it” memory management style that I did in lecture (rather than the “have many multiple pointers to each link object” style, which works well with garbage collection but typically generates garbage in C++). Note that, except for the generation of garbage and the time required for various operations, these styles (and reference counting, below) are equivalent as long as there are no list mutator operations — in other words, switching from one implementation to another can affect the time or memory needed, but nothing else about the outcome of a program.

Do the following exercises with this project:

1. Complete the function `small_test` (right above `main`), replacing the parts that say `__FILL_THIS_IN__` to show the number of “links” that you would expect to exist if each list object has its own set of links. Then run the program and check to see if your predictions are right (feel free to adjust them if you wish).
2. Set the `LIST_DEBUG` constant in `list.h` to be `true` and run your `small_test` function and none of the other tests (by leaving `just_do_small_test` as `true` in `main`). Draw (using `gimp`, `inkscape`, or some other program) a diagram of the allocated memory in `small_test` just before the creation of `p` and the call to `function`. Save your drawing (and the one below) as a PDF file in your project.
3. Draw another diagram showing the memory allocation diagram for `small_test`, but imagining that you have implemented a reference-counting garbage collector for lists: list cells should not be copied, and each cell should be marked with a count.

**Submit what you have at this point — you have reached the “basic credit” level.**

4. Change the linked list class to use reference counting. (Since the list is acyclic, this strategy will always detect any garbage.) Your classes should ensure that no unnecessary copies of link objects are made and neither dangling references nor garbage can be created by any program using linked lists.
5. Now re-run your small test and confirm (by looking at the addresses) that the diagram you drew in Question 3 is correct (no need to hand anything in for this).
6. Finally, run all the tests, and see if it consistently allocates and de-allocates links (there will be a failed assertion if it does not).

Remember to submit your project when you’re done.