

Garbage-Collected Pointers

In this assignment, you will develop a template class that associates a reference count with a generic pointer in order to enable automatic garbage collection. The objective of the assignment is to give you some familiarity with C++ templates as well as basic experience with the reference-counting approach to garbage collection, profiling code, and understanding tradeoffs such as BFS/DFS. I've also thrown in a few questions about random graphs and bugs as thought problems.

A starting package that will use your template is available from the class web site. Two files are included: `sanity.cc` and `path.cc`. In contrast with my usual philosophy on labs, I'll declare in advance that I'll only use these two files to test your code. There's at least one subtlety that will take some thinking anyway. The `sanity.cc` code is a set of checks designed to ensure that your implementation does not contain any obvious errors. The `path.cc` code is a code that performs experiments on randomized graph connectivity. It generates a 2D lattice of nodes, connects each node to the node to its right and to the node below it with some specified probability, and then calculates statistics on which of the diagonal lattice elements can be reached. By repeatedly generating random graphs and adding up the reachability stats for each graph, the program estimates the probability that each diagonal element can be reached in a random graph instance. In addition to being an interesting probability problem for you to think about, it exercises your garbage-collection code in a reasonable way. By creating and destroying many small or medium graphs, you can easily see if your implementation leaks memory.

Specifics

Here are a few details on the things that you'll need to do:

- Your template **must** be called `gcp` (garbage-collected pointer) and must take a single type as an argument, which will represent the type of pointer to be garbage collected. Write your template in a file called `gcp.h`.
- Each `gcp` instance should be associated with a pointer and a count of how many times that pointer is referenced by `gcp` instances.
- You must implement a constructor with no arguments as well as a constructor operating on a pointer to the template class. One acceptable approach is to use a default argument of 0 (NULL) for a single-argument constructor.
- You must implement the dereference operator `->`. Stroustrup has a discussion on “smart pointers,” if you need hints as to how to make this operator work.
- You must implement equality and inequality comparison operators, either as methods or friends (either should work with the code that you're given, I think).
- The total implementation is only about 60 lines of code, not counting header comments, *etc.*, so...
- Do this lab on your own (no teams).

Experiments

Here are a couple of problems that you should think about with the `path.cc` code. Turn in your answers with your code.

- If you run a large number of small graphs (say 10×10 with 50% connectivity), you'll notice that the probabilities do not form a geometric sequence. Explain why not.
- Look for the word "EXPERIMENT" (all caps) in the `path.cc` file. Changing a variable from a `gcp` to a `gcp` reference changes the statistics; again, try comparing 10×10 graphs with 50% connectivity. Which one is right? Explain the bug with using the other version. *Note: I actually made this mistake, and it took me a while to track it down. You have to think of `gcp`'s as pointers to use them correctly.*
- Compile and run something reasonably long (a few minutes or more) and report what fraction of time is spent in each of your template routines. You'll need to turn off optimization here, as even the lowest level will inline all of your template code into the `path.cc` routines.
- Compare BFS and DFS (see the code to find how to change) using one trial of a 500×500 grid with 100% connectivity. Lack of optimization will obscure this difference; use `-O9` for best results. You won't see much difference. Explain why. Change the line near "MARK" in the code and try again using 15×15 and 100% connectivity. Explain why the difference is more substantial.
- For small probabilities, the likelihood of connecting to nodes further down a diagonal decreases rapidly (try 20×20 for this one). However, for large probabilities, the probability becomes almost flat (it's roughly the same for elements 15-19, for example). Find the percentage at which this phase shift occurs and explain why it occurs around that probability. *Note: This problem is challenging.*

Handin

E-mail me your `gcp.h` code and answers to the experiment questions by the due date. Be sure that you test your code with unmodified versions of `sanity.cc` and `path.cc`. You'll lose some points for compiler warnings, and most points for failing to compile. If you want to send me `gprof` output for your profiling results, be sure to trim it down and highlight the parts corresponding to your template class.