15-411 Compiler Design, Fall 2014
Lab 6 - Garbage Collection

Instructor: Frank Pfenning
TAs: Flávio Cruz, Tae Gyun Kim, Rokhini Prabhu, Max Serrano

Compilers due 11:59pm, Thursday, December 4, 2014
Papers due 11:59pm, Tuesday, December 9, 2014

1 Introduction

The main goal of the lab is to explore advanced aspects of compilation. This writeup describes the option of implementing garbage collection; other writeups detail the option of implementing optimizations or retargeting the compiler. The language L4 does not change for this lab and remains the same as in Labs 4 and 5.

2 Requirements

You are required to hand in three separate items:

1. The working compiler and runtime system that implement optimizing transformations,
2. a testing framework, and
3. a term paper describing and critically evaluating your project.

2.1 Compilers

Your compilers should treat the language L4 as in Labs 4 and 5. Regardless of what flag is passed to your compiler, you should only implement safe memeory semantics for the purpose of this assignment.

Garbage Collection

You have complete freedom which kind of garbage collector to implement. A garbage collector will consist of the compiler proper and the runtime system. The interface from the compiled code to the runtime system should be part of your design. Reasonable choices are a mark-and-sweep or a copying collector, but even a conservative collector is acceptable. Incremental collectors are significantly harder and should only be attempted if you already have a basic collector working.

Grading criteria includes:

1. Functional correctness is paramount. You should not mutate the heap in such a way as to result in the incorrect execution of programs. Your compiler should continue to function correctly, despite any changes to the binary interface you use to interface with the garbage collector.

2. The absence of memory leaks comes second. A garbage collector that takes a whole lot of processor time is not of much use if it cannot effectively collect parts of the heap that are no longer referenced.

3. Performance is a distant third, and has a very minor effect on your grade. Optimizations to garbage collectors require a significant amount of time. Therefore, we recommend that you avoid premature optimizations.

2.2 Tests and Measurement Tools

You need to demonstrate that your garbage collector does not corrupt the heap, and does not leak memory. To this end, feel free to search through all of the test cases that we have accumulated through this semester for programs that are both realistic and usefully contrived to assemble a test suite. You will need to write contrived test cases designed specifically to ensure that your garbage collector goes through several collection cycles without corrupting the heap or leaking any memory. This means you may need to run your compiler in an environment where memory is artifically constrained. You will be graded on how well you test your garbage collector.

2.3 Term Paper

Your paper should follow this outline.

1. Introduction. This should provide an overview of your implementation and briefly summarize the results you obtained.

2. Compilation. Describe the data structures, code, and information generated by the compiler in order to support the garbage collector.

3. Runtime System. Describe the runtime system of the garbage collector, giving details of the algorithms and also its implementation (most likely in C).

4. Testing Methodology. Describe the criteria based on which you selected and designed your tests, and explain how you use them to verify the functionality of your garbage collector.

5. Analysis. Critically evaluate your collector and sketch future improvements one might make to its basic design.

The term paper will be graded. There is no hard limit on the number of pages, but we expect that you will have approximately 5-10 pages of reasonably concise and interesting analysis to present.

3 Deadlines and Deliverables

3.1 Compiler Files (due 11:59pm on Thu Dec 4)

There is no plan to automatically grade your compilers on Autolab, although we will make sure it builds correctly. You will receive a nominatl score of 0 if your compiler does not build and 1 if it does. All files should be collected in a directory `compiler/` which should contain a `Makefile`. **Important:** You should also update the `README` file and insert a roadmap to your code. This will be a helpful guide for the grader.

Issuing the shell command

```
% make l4c
```

should generate the appropriate files so that

```
% bin/l4c --safe <args>
```

will run your L4 compiler in safe mode with support for garbage collection. The command

```
% make clean
```

should remove all binaries, heaps, and other generated files.

Furthermore, if necessary you should modify the driver from Lab 4 or 5 so that tests can suitably be linked against the garbage collector. Your garbage collector will likely be a library that is to be dynamically linked against compiled binaries – and your driver should take care to set the path for dynamic linking, etc. If there are any special instructions we need to follow in order to be able to run the driver on your compiler and test it, specify these instructions in your README file.

All your material must be committed into `lab6gc` in the same way that you submitted your compiler in previous assignments.

3.2 Tests and Measurement Tools (due 11:59pm on Thu Dec 4)

In a directory called `tests/`, include all the tests that you selected or wrote for the purpose of testing your garbage collector. If they are to be used in a different way than a vanilla L4 test, you should include a README file explaining exactly how to use your tests.

3.3 Term Paper (due 11:59pm on Tue Dec 9)

Submit your term paper in PDF form via Autolab before the stated deadline. Early submissions are much appreciated since it lessens the grading load of the course staff near the end of the semester. **You may not use any late days on the term paper describing your implementation of Lab 6!**

4 Notes and Hints

- Limit optimizations. Garbage collection is easier if fewer optimization are applied to the code, especially where memory references are concerned. In order to concentrate on the garbage collector it is probably a good idea to stay away from optimizations altogether.

- Apply regression testing. It is very easy to get caught up in the new functionality.
- Copying vs. mark-and-sweep collector. Experience in previous years indicates that a copying collector is easier to implement for our language than a mark-and-sweep collector because the data structures are simpler.