15-411 Compiler Design: Lab 6 - Garbage Collection
Fall 2009

Instructor: Frank Pfenning
TAs: Ruy Ley-Wild and Miguel Silva

Compilers due: 11:59pm, Thursday, December 3, 2009
Term Paper due: 11:59pm, Thursday, December 10, 2009

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 two separate items: (1) the working compiler and runtime system, and (2) a term paper describing and critically evaluating your project.

3 Benchmarks and Tests

We will not test the garbage collector for efficiency, only for correctness.

The format of the test files is as for Lab 5, except each should export a function `int test(param : int)`. The memory usage of the program should go up with the parameter.

The test files also supply a function `main` which calls `test` after seeding the random number generator with a fixed number. The output of this function should be the expected return value in the first line.

The driver may call `test` multiple times with different parameters, random seeds, and memory limits to verify that the program does not crash. It will also call `main` to verify the correctness of the computation.

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.

Similarly, you should avoid optimizations; efficiency is only a minor concern for this lab. Functional correctness is paramount, and the absence of memory leaks is a secondary criterion.

4 Compilers

Your compilers should treat the language *L4* as in Labs 4 and 5, including `extern` declarations. For the garbage collector, only safe compilation should be used, no matter what flag is passed in.

5 Project Requirements

On the Autolab server, the hand-in and status pages for the optimization and garbage collection projects are separated, since different drivers will be employed.

Compiler Files (due 11:59pm on Thu Dec 3)

As for all labs, the files comprising the compiler itself 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 modes, although as noted above you may choose to ignore the flag.

The command

```
% make clean
```

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

Using the Subversion Repository

The handout files for this course can be checked out from our subversion repository via

```
% svn checkout https://cvs.concert.cs.cmu.edu/15411-f09/<team>
```

where `<team>` is the name of your team. You will find materials for this lab in the `lab6gc` subdirectory. Or, if you have checked out `15411-f09/<team>` directory before, you can issue the command `svn update` in that directory.

After first adding (with `svn add` or `svn copy` from a previous lab) and committing your handin directory (with `svn commit`) to the repository you can hand in your tests or compiler by selecting

```
S3 - Autograde your code in svn repository
```

from the Autolab server menu. It will perform one of

```
% svn checkout https://cvs.concert.cs.cmu.edu/15-411/<team>/lab6gc/compiler
```

to obtain the files directories to autograde, depending on whether you are handing in your test files or your compiler.

If you are submitting multiple versions, please remember to commit your changes to the repository before asking the Autolab server to grade them! And please do not include any compiled files or binaries in the repository!

Term Paper (due 11:59 on Thu Dec 10)

You need to describe your implemented compiler and critically evaluate it in a term paper of about 10 pages. You may use more space if you need it. The recommended outline varies depending on your project. Submit a file `<team>-gc.pdf` via email to the instructor at `fp@cs`.

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

6 Notes and Hints

- Limit optimizations. Garbage collection is easier if fewere 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. Besides the benchmarks we have a large test suite collected over several labs; use these for regression testing to make sure your compiler remains correct.

- Copying vs. mark-and-sweep collector. Experience last year indicates a copying collector is easier to implement for our language than a mark-and-sweep collector because the data structures are simpler.