15-411 Compiler Design, Fall 2014
Lab 6 - Create Your Own Adventure

Rob Simmons, Will Crichton, Grant Della Silva, Matt Bryant, Anshu Bansal

Proposal due 11:00pm, Thursday, December 3, 2015
Compilers due 11:00pm, Thursday, December 10, 2015
Papers due 11:00pm, Tuesday, December 15, 2015

1 Introduction

The main goal of the lab is to explore advanced aspects of compilation. This writeup describes the process for proposing your own Lab 6; other writeups detail other options.

2 Requirements

You are required to hand in three separate items:

- The working compiler and support materials (runtime, etc.) that implement your proposed project.
- Any additional tests and framework to test your project.
- A term paper describing and critically evaluating your project.

2.1 Possibilities

This is your opportunity to be creative! Any substantial and interesting extension or modification to your compiler that welcome. The pre-suggested options for Lab 6 should give you an idea of the expected scope of the project. As inspirational material, other possibilities for Lab 6 might look like:

- Extending the L4 language in some *significant* interesting way (parametric polymorphism, exceptions, concurrency, parallelism, ...).

  (Update Nov 19: first-class were removed from the list above because we covered closures in some detail in class. Parametric polymorphism is still on the list, because as we discussed in class, it adds significant complexity to either box and unbox values or to do monomorphization.)
- Retargeting your compiler to another hardware instruction set (without LLVM, otherwise that's an LLVM project).
- Retargeting your compiler to some interesting other programming language (Verilog, dc, Malbolge, ...).

2.2 Tests and Measurement Tools

You need to demonstrate that your compiler is correct. How you do this will obviously depend heavily on the project you propose. You may need to write new tests to exercise the Lab 6 portions of your compiler; you may need to construct a customized testing framework. Before proposing a project, you should give some thought to your testing approach. If there will be no way to check whether your compiler meets the goals of your project, we will probably not be satisfied with your project!

2.3 Term Paper

Your paper should follow this outline.

1. Introduction. This should provide an overview of the proposed project, give a sketch of your implementation, and briefly summarize results.

2. Project. Give a specification for your project.

3. Implementation. Describe the modifications made to your compiler to meet the project goals, including data structures and algorithms. Describe also any runtime system required for your project.

4. Testing Methodology. Describe the design of your testing approach. Include any relevant information such as the criteria you used as you selected or designed your tests, how you constructed your testing system, how your testing approach verifies the functionality of your compiler. What you put in this section will, of course, depend on your project.

5. Analysis. Critically evaluate your compiler and sketch future improvements one might make to your current implementation.

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

All your code should be placed in subdirectories of the `lab6cyoa` directory as before. The autograder will run regression tests against your own tests and the same tests it used in Lab 5, but these will *not* directly contribute to your grade (in particular, you may intentionally implement features that are not backwards-compatible). We will grade you based on the code and `README` file(s) you have checked in at the deadline.

3.1 Proposal (due 11:00pm on Thu Dec 3)

The exact timing of this isn't terribly important, but by the morning of Friday, December 3, you should have emailed the professor (`rjsimmon@cs`) a one-page PDF that describes what your specific goals for the project are and how you will determine whether or not you have met them.

3.2 Compiler Files (due 11:00pm on Thu Dec 10)

The files comprising the compiler should be collected in a directory `compiler/` which should contain a `Makefile`. **Important:** You should also update the `README` file with a roadmap to your implementation. This will be a helpful guide for the grader.

Issuing the shell command

```
% make
```

should generate the appropriate files so that

```
% bin/c0c --exe <args>
```

will run your compiler and produce an executable. It is not necessary to continue supporting any compiler flags besides `-t`.

After running `make`, issuing the shell command

```
% make test
```

should run your own tests and print out informative output. The command

```
% make clean
```

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

If it is reasonable, you should modify the driver from lab 5 to test your extended compiler. 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.

Runtime environment (due 11:00pm on Thu Dec 10)

Because you may have implemented a new runtime, your compiler should have an additional flag `--exe`. If your compiler is given a well-formed input file `foo.l1` or `foo.l2` as a command-line argument and is also given the `--exe` argument, it should generate a target file called `foo.l1.s` or `foo.l2.s` (respectively) in the same directory as the source file, and should *also* compile the runtime and link it with your generated assembly to create an executable `foo.l1.exe` or `foo.l2.exe` (respectively). We will test your compiler against the regression suite, but the grade reported by the autograder won't directly contribute to your grade.

3.3 Tests and Measurement Tools (due 11:00pm on Thu Dec 10)

In a directory called `tests/`, include all the tests that you selected or wrote for the purpose of testing your project. 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, and `make test` should run your tests.

If you also do any performance testing in the same vein as `lab5`, include the necessary files in `bench/`.

3.4 Term Paper (due 11:00pm on Tue Dec 15)

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

- Discuss your ideas with the course staff to get feedback on feasibility and scope of the project before embarking on it.
- Try to identify some intermediate goals in case your overall project turns out to be too ambitious. You should clearly indicate what your intermediate goals are in your proposal.
- Apply regression testing. It is very easy to get caught up in new features. Please make sure that the L4 portion of your compiler continues to work correctly (keeping in mind that not all changes are expected to be backward compatible)!