15-411 Compiler Design, Spring 2024
Lab 6 - Create Your Own Adventure

Jan and co.

Due 11:59pm, Saturday, May 4th, 2024

1 Introduction

The main goal of Lab 6 is to explore advanced aspects of compilation, and for CYOA that typically means implementing your written 5 project proposal (see written 5 handout for details). Hopefully you have had time to think through the implementation and to ensure that your ideas are feasible. As we mentioned, you do not have to stick to the ideas in your proposal, so feel free to choose another pre-defined L6 options or maybe a different CYOA altogether. However, do let us know if you make this decision down the line so the course staff knows accordingly and can properly advise you if needed.

2 Deliverables

2.1 Compiler

You will need to submit all the files associated with your working compiler. 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).

Issuing the shell command

```
% make
```

should generate the appropriate files so that

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

will run your compiler and produce an executable. You may stop supporting all other compiler flags except `-t`. This means you no longer need to support the `--unsafe` flag or any optimization flags for this assignment.

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. Please specify in your README any special instructions we need to follow in order to be able to run the driver on your compiler.

You should also update the `README` file and insert a roadmap to your code. This will be a helpful guide for the grader. In particular, since there are likely to be many different projects undertaken, do introduce the project at the very top

of the README.

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!

Please submit test cases and any associated testing framework in a clearly marked directory, such as `lab6-tests/`. 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/`.

2.3 Term Paper

Your term paper should describe and critically evaluate your project, following the outline below.

1. Introduction: Provide an overview of the proposed project, give a sketch of your implementation, and briefly summarize results.
2. Specification: 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: 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, and how your testing approach verifies the functionality of your compiler.
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.

Submit your term paper in PDF form via Gradescope 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!

2.4 Project Rubric

To fairly evaluate you, we need to know what you are planning to do. You are expected to submit a tentative grading rubric by **Wednesday, April 17th 2024, at 12:00PM Noon**. This is a Google Form which you can find here (and on Piazza).

This form should include a description of the project (which you can adapt from your proposal), what an 80% can look like for your project, and the extra features on top of an 80% that define what a 100% looks like for your project. **In particular, please make note of any changes if your project has deviated from your L6 proposal.**

You will receive feedback from course staff about your rubric should we deem your rubric to be too lenient or too harsh, but we will inform you of any needed modifications. Your final grade will be determined by the quality of your implementation, your term paper, and the extent to which you meet the goals you have set in your proposal.

Note that the rubric serves as a guideline for your project, but is not a guarantee of the grade you will receive. This is primarily a chance for you to calibrate your project expectations with those of the course.

Some sample projects are as follows:

2.4.1 New Architecture Target Language

Description: This project aims target a new assembly language (ARM, RISC-V, etc.) instead of x86. We aim to generate code for this new architecture, test it, and potentially implement an optimization specific to this architecture.

Milestones for 80%:

- Fully working translation from L4 to the new assembly language.
- A test suite for the new assembly language that proves the validity of the translation (can modify / make use of existing test suite).
- Construct the means to test the new architecture and benchmark its performance characteristics against x86-64.

Additional Milestones Required for 100%:

- A new optimization that is specific to the new architecture i.e. hyperpipelining, branch-prediction, instruction selection heuristics, etc.
- Analysis of the optimization and how it improves performance in specific use-cases.

2.4.2 Higher Order Functions

Description: This project aims to implement higher order functions in the L4 language. This will require changes to the parser, type checker, internal representations, and code generation. We will make use of lambda functions as an abstraction, and implement a typechecker that can detect function types.

Milestones for 80%:

- Working typechecker for detecting function types.
- Working front-end for parsing lambda functions as well as higher order types.
- Working internal representation for recursively nested lambda functions.
- Working code-generation for lambda functions.
- A test suite of at least 15 tests that proves the validity of the implementation.

Additional Milestones Required for 100%:

- Support local variable captures in lambda functions i.e. closures.
- Support for recursive variable shadowing in addition to accurate determination of scope for captured lambda functions.
- A test suite of at least 10 tests that proves the validity of higher-order recursively nested lambda functions.

3 Notes and Hints

- You will be formally defining the scope of your project, as well as the goals and milestones for your project in a grading proposal. This proposal should provide you meaningful milestones in case your overall project turns out to be too ambitious.
- Your project definition will be revised and sent back to you to act as a set of guiding milestones for your project. Please make sure to check your email, and reach out if you have any questions.
- 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)!