# COMP 506, Spring 2018
# An Optimizer for ILOC[1]

Due Date: **April 20**, 2018 at 11:59 PM

## 1 Introduction

This document describes the programming assignment for COMP 506 at Rice University in the Spring Semester of 2018. For the project, you will build an optimizer that tries improve the running time of a set of *test codes* across a set of *test inputs*. Test codes and test inputs will be available on CLEAR.

All test codes will be presented in a subset of the **ILOC** intermediate form. The specific subset is described in § 6.1 of the **ILOC** Simulator document, available on the Projects page of the COMP 506 web site. Your optimizer will need to:

- ⇒ Scan and parse the **ILOC** code
- ⇒ Build an internal representation for the **ILOC** code
- ⇒ Perform two optimizations.
- ⇒ Print the transformed **ILOC** code in a form that can be read and executed by the **ILOC** simulator

The executable for your optimizer should be named `opt`. It should accept a command line of the form:

`opt` *flags* `file.i`

where *flags* specify the optimizations to run (and the order in which to runt them, and `file.i` is the name of the **ILOC** input file that will be optimized. Your optimizer should write its output file to `stdout`.

Your optimizer should implement two of the three following optimizations:

1. *value numbering*: local value numbering or superlocal value numbering; specified with a `-v` flag
2. *loop unrolling*: unroll inner loops by a factor of four; specified with a `-u` flag
3. *loop-invariant code motion*: find computations that are invariant in inner loops and move them to a place where they execute less often; specified with a `-i` flag

If someone invokes `opt` with a flag for an optimization that you did not implement, `opt` should print out a suitable error message, such as

```
% opt -v -i file.i
-i:  code motion not implemented
```

To be clear, you must implement two of the three optimizations. Your optimizer should handle the command line for any unimplemented optimization gracefully.

[1] **ILOC** is an intermediate representation described in Appendix A of *Engineering a Compiler* [1]. The specific **ILOC** subset used in this project is described in the documentation for the **ILOC** simulator, which is available on the course web site: `http://www.clear.rice.edu/comp506`

The order in which the optimization switches appear should determine the order in which the optimizations apply. For example,

```
opt -v -i file.i
```

should optimize the file `file.i`, applying value numbering followed by loop-invariant code motion, while the command line:

```
opt -v -i file.i
```

would apply loop-invariant code motion before value numbering.

To measure the effectiveness of your optimzer, you will use the **ILOC** simulator to measure the number of cycles that it takes for (1) the original code, (2) the code after applying each optimization by itself, and (3) the code after applying the two optimizations in each order.[2] We will provide a small set of test input files, in the `~comp506/students/lab3/TestCodes` directory on CLEAR. You will report your experience in a brief written report, based on a form that we will provide.

## 2 Rules

As in any programming assignment in a for-credit course, there are a number of rules that you must obey. The intent of these rules is to create a fair environment and to remove any ambiguity in the assignment.

1. You are expected to do your own implementation work. You are encouraged to discuss the project with your classmates, your advisors, the professor, and any other person whom you believe may have insight. However, the implementation must be your work.
2. Your optimizer must run on the CLEAR system. You can develop and debug it on any system available to you. It must, however, run correctly on CLEAR.
3. You are expected to use the C programming language, The runtime speed of your optimizer is not a consideration in grading; it should, however, not be embarassingly slow.

   An optimizer that takes more than a couple of minutes to process any of the examples would be considered embarassingly slow. If your optimizer is embarassingly slow, expect to explain why it is slow in your report.
4. The primary goal of your optimizer is to reduce the number of cycles that the simulator reports when the input codes are executed. That is, labs will be compared based on the number of cycles that it takes to execute the optimized code that they produce.

## 3 Software

The **ILOC** simulator is available in `~comp506/students/lab3/Sim` on the CLEAR systems. The directory should include both source and executable versions. The simulator documentation is available on the web page, as well as in `students/lab3/Sim`. Section 2 of the

[2]You will perform a total of five tests on each program: orginal code, code with optimization $x$, code with optimization $y$, code with the sequence $xy$, and code with the sequence $yx$.

simulator document explains the available command-line options for the simulator, including the `-d` option that allows you to specify a test input.

## 4 Deadlines

- Your code and report are due on **April 20**, 2018 at 11:59 PM.
- Directions for submitting the code and the report will be posted to the class discussion board on Pizza.
- Labs submitted after April 27, 2018 will have a late penalty. The lab's grade will be multiplied by (1 - $0.20^n$), where $n$ is the number of days late, rounded up to an integer.

## 5 Questions

Post questions to the class discussion board on Piazza.

## References

[1] Keith Cooper and Linda Torczon. *Engineering A Compiler.* Elsevier Morgan-Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2011.