The course project is to build a compiler for a small language. This is a "living" document will be revised throughout the semester until it is a complete, if sometimes informal, language specification. Revisions may include additions, removals and changes to meet pedagogical goals and to ensure internal consistency.

(X) means zero or one occurrence of X { X } + means one or more occurrences of X { X } * means zero or more occurrences of X

SECTION 1: Lexical structure (see version 1.1 of this document)
SECTION 2: Syntactic structure (see version 2.0 of this document)
SECTION 3: Type checking and semantics (see version 3.0 of this document)
SECTION 4: Intermediate code generation (see version 3.0 of this document)
SECTION 5: Assembly code generation (see version 4.0 of this document)
SECTION 6: Machine-independent optimizations (this document)

At this stage of the project the main goal is to implement various code optimizations from chapters 8 and 9.

Section 8.5 discusses *local* optimizations, optimizations which take place within basic blocks[1]:

8.5.2 – p. 534 – local common subexpression elimination (1 point) with flag: -opt 1
8.5.3 – p. 535 – (local) dead code elimination (1 point) with flag: -opt 2
8.5.4 – p. 536 – arithmetic identities, reduction in strength, and constant folding (1 point) with flag: -opt 4

Section 8.7 discusses *peephole* optimizations, which can be applied to intermediate or target code. For our purposes apply any peephole optimizations to the intermediate code:

8.7.1 – p. 550 – eliminating redundant loads and stores (2 points) with flag: -opt 8
8.7.2 – p. 550 – eliminating unreachable code (2 points) with flag: -opt 16
8.7.3 – p. 551 – flow-of-control optimizations (2 points) with flag: -opt 32

Chapter 9 covers a variety of machine-independent optimizations not restricted to basic blocks:

9.1.4 – p. 588 – global common subexpression elimination (2 points) with flag -opt 64
9.1.5 – p. 590 – copy propagation (2 points) with flag -opt 128
9.1.6 – p. 591 – (global) dead code elimination (2 points) with flag -opt 256
9.1.7 – p. 592 – code motion (2 points) with flag -opt 512
9.1.8 – p. 592 – induction variables and reduction in strength (3 points) with flag -opt 1024

Optimizations can be combined: *-opt 1025* implies both *-opt 1* and *-opt 1024*. Similarly, *-opt 7* implies *-opt 1*, *-opt 2*, and *-opt 4*. If an optimization is specified that is not supported it must be ignored. Thus, *-opt 2047* must apply all supported optimizations.

Your compiler must also support the flags *-opt grading* and *-opt supported*. If either of these options is specified then all other options are ignored, and no compilation occurs. Instead, *-opt grading* must print *X* where *-opt X* will run all and only those optimizations that together count for 7 points which you want us to grade. On the other hand, *-opt supported* must print *Y* where *-opt Y* will run all implemented optimizations.

Your submission will be graded out of 7 points. If *-opt grading* prints a value for optimizations which together count for more than 7 points we will grade our choice of optimizations which add up to 7 points (or

[1] Be mindful of the discussion of 8.5.5 (array references), which is relevant to our language, and in general also 8.5.6 (pointers), which is not.

8, in case you did four 2 pointers), even if those are not your best-scoring optimizations. Points earned above 7 do not benefit you.

Grading

In addition to assessing the optimizations your compiler performs we will re-assess the functionality expected in earlier submissions (except PR01, since all scored 100% on that). Each submission focused on a specific part of the overall compiler:

PR01 - lexical analysis (LA)
PR02 - syntax analysis (SA)
PR03 - type checking and intermediate code generation (IC)
PR04 - assembly code generation (AC)
PR05 – machine-independent optimizations (OP)

Your overall project grade will be determined according to the following:

grade for LA = PR01 grade
grade for SA = max(PR02 grade, SA re-assessment in PR05)
grade for IC = max(PR03 grade, IC re-assessment in PR05)
grade for AC = max(PR04 grade, AC re-assessment in PR05)
grade for OP = PR05 grade

The overall project grade will be the sum of the grades for LA, SA, IC, AC and OP, weighted evenly.

SUBMISSION & GRADING:
Submit your code using Autolab. Submissions are due no later than 5:00 PM on Monday May 14.