COM S 440/540 Project part 5

Compiler — code generation (2)

1 Basic requirements

You should have a working compiler, that is able to parse and type check a single input program written in our subset of C, and produce an equivalent program in our target language (see the stack machine document). Your compiler must now generate correct code for (possibly nested) branching statements and loops. Additionally, you must display appropriate error messages (including the file name and line number where the error occurs) to standard error, for lexer errors (cannot open files, bad tokens, unclosed comments), syntax errors, type mismatch errors, and other errors. After the first error, you may make a "best effort" to continue processing the input file, or exit. Your executable should be named `compile`, and code generation should be invoked using the "`-i`" switch. Generated code should be written to standard output. For example,

```
prompt$ ./compile -i infile.c > infile.ir
```

should produce an executable file, in the file format specified in the stack VM document, that can be run on the stack virtual machine. If you wish, you may include extra information to help with debugging, as comments in the executable file.

2 Grading

For all students: implement as many or as few features listed below as you wish, but keep in mind that some features will make testing your code *much* easier, and a deficit of points will impact your overall grade. Excess points will count as extra credit.

For code generation "without short circuiting", your compilers will be tested using integer variables for the condition. The basic tests for each construct are as follows, where `x` is an integer variable.

```
if (x) { /* statements */ }

if (x) { /* statements */ } else { /* statements */ }

while (x) { /* statements */ }

do { /* statements */ } while (x);

for (/* initialize */; x; /* update */) { /* statements */ }
```

More advanced tests will use a single comparison as the condition.

Tests for short-circuiting boolean expressions will call functions that output characters (so make sure those are working) as part of the boolean expression, to make sure the expression short circuits.

Ease of building the executable	10 points
Developer documentation (`developers.tex`)	20 points

In addition to the documentation for all previous parts, this should discuss the critical data structures used in your compiler, and generally how the target code is generated. Since your target audience is other developers who have taken this course, you may refer to sections of the textbook or material discussed in lectures.

Non-instruction output	10 points

Your compiler correctly generates the constant, globals, number of functions, and function header / trailer portion of the target executable. That is, all portions of the target executable except instructions are correctly generated.

Expressions and function calls	10 points

This includes the most basic functionality from the previous part of the project, namely assignments to variables and function calls, that will be necessary to test this part of the project.

Without short circuiting

if-then	5 points
if-then-else	5 points
while	10 points
do-while	10 points
for	10 points
`?:`	5 points
`break` (requires a working loop)	5 points
`continue` (requires a working loop)	5 points
Comparisons: `==`, `!=`, `>`, `>=`, `<`, `<=`	10 points

Short circuiting

And, Or, Not	5 points
Comparisons	5 points
Boolean assignments	5 points
if-then, if-then-else, `?:`	5 points
while, do-while	5 points
for	5 points

Total points for students in 440 (any excess earned is extra credit)	100 points
Total points for students in 540 (any excess earned is extra credit)	120 points