CSE 5343, Programming Project 1: Scanner for simpleC
Due Tuesday, January 23, 11:59 pm (30 points)

The goal of this project is to extend an existing scanner for a simple subset of C. We will refer to this language as simpleC. The language will be used in all projects this semester.

Use the setup from Project 0. Read readme.txt, Scanner.jflex, Parser.cup, Compiler.java, and all classes in package ast. *It is essential to do this reading early and to ask any clarification questions as soon as possible. In particular, if you do not have experience with object-oriented programming in Java, please proactively reach out to me for clarifications if necessary.*

Goals

Generalize numeric literals. In simpleC, the only primitive types are int and double. The provided scanner uses simple patterns to define literals from those types. Generalize the scanner to recognize (some of) the literals from the real C language, as described in TODO comments in Scanner.jflex. Use the C language spec document available under Resources on the course website (Sections 6.4.4.1 and 6.4.4.2).

Generalize identifiers. Change the scanner to recognize general identifiers, as defined by the C spec (Section 6.4.2.1). See the TODO comment in Scanner.jflex for details.

Generalize binary operators. Currently the implementation handles binary operators + (add), * (multiple), and = (assign). Modify Scanner.jflex and Parser.cup to also handle binary operators – (subtract), / (divide), and % (modulo). In addition, add handling for the following compound assignment operators (Section 6.5.16.2): +=, -=, *=, /=, and %=. Make sure that precedence and associativity for these 8 new operators are specified correctly in Parser.cup (see https://www2.cs.tum.edu/projects/cup/docs.php#precedence for details). Add parser actions to generate the corresponding nodes in the abstract syntax tree (AST). Modify ast/BinaryExpr.java to handle these new operators.

Details

1) The input will always be ASCII – you do not need to worry about Unicode characters (that is, you can ignore *universal-character-name* defined in Section 6.4.3 of the ANSI C document).

2) You can assume that each numeric literal has a value small enough to fit in the corresponding Java type.

3) Semantic checks: There are various *semantic checks* that would normally happen after parsing. For example, each identifier occurring inside an expression must be in the scope of some declaration, and the uses of the identifier should be consistent with its declared type. As another example, the left operand of the assignment operator cannot be a constant value (i.e., we cannot have **4=5** as a valid expression). Such constraints are not expressed via the context-free grammar of the language, and therefore should **not** be your concern when performing parsing in Project 1 (and the subsequent Project 2). In essence, you can assume that any input

program is guaranteed to pass all such semantic checks after parsing, but we will **not** actually do the checking.

Testing

Write many test cases and test your implementation with them. Submit at least 5 test cases with your submission. The test cases you submit will not affect your score for the project. Put them in the same location as the provided file t1.c and name them t2.c, …

Submission

After completing your project, do
cd p1
make clean
cd ..
tar -cvzf p1.tar.gz p1
Then submit p1.tar.gz in Carmen.

General rules (copied from the course syllabus)

Your submissions must be uploaded via Carmen by midnight on the due date. The projects must compile and run on **stdlinux**. Some students prefer to implement the projects on a different machine, and then port them to stdlinux. If you decide to use a different machine, it is entirely your responsibility to make the code compile and run correctly on stdlinux before the deadline. In the past many students have tried to port to stdlinux too close to the deadline, leading to last-minute problems and missed deadlines.

Projects should be done independently. General high-level discussion of projects with other students in the class is allowed, but **you must do all design, programming, testing, and debugging independently.** Projects that show excessive similarities will be taken as evidence of cheating and dealt with accordingly. Code plagiarism tools may be used to detect cheating. See the syllabus under "Academic Integrity".

The projects are due by 11:59 pm on the due day. You can submit up to 24 hours after the deadline; if you do so, your score will be reduced by 10%. **ONLY THE LAST SUBMITTED VERSION WILL BE CONSIDERED. Triple-check carefully that you have submitted the correct version. If you submit the wrong version of your code, and you get a low score (or zero score), I will NOT consider resubmissions – the original low/zero score will be assigned WITHOUT DISCUSSION.**

If you submit more than 24 hours after the deadline, the submission will not be accepted. NO EXCEPTIONS TO THIS RULE WILL BE CONSIDERED. NO REQUESTS FOR RESUBMISSION WILL BE CONSIDERED. MAKE SURE YOU SUBMIT THE CORRECT CODE VERSION.

Read the project description **very carefully, several times, start-to-end.** If you need any clarifications, contact me immediately (do **not** wait until the last minute). **Test extensively.**

Accommodations for sickness and other special circumstances will be made based on university guidelines. Please contact me **ahead of time** to arrange for such accommodations.