CSE 5343, Programming Project 2: Parser for simpleC
Due Friday, February 2, 11:59 pm (30 points)

The goal of this project is to extend your implementation from Project 1 with more language features: while loops, for loops, and restricted boolean expressions. Create a directory `p2` under `proj`. Copy your Project 1 to `p2` and start from there. Do all work for Project 2 in `p2`.

Goals

Generalize statements. The implementation handles if-then and if-then-else statements. Add handling of *while loops* (Sec 6.8.5 of the C spec). Modify the parser, the scanner, and the AST accordingly. As part of AST modifications, create a class WhileStmt similar to class IfStmt.

Similarly, add handling of *for loops* (Sec 6.8.5), but only for the first form—the one **without** a declaration. For simplicity, in simpleC we will require that all three expressions in the loop are present. Your parser should enforce this restriction. For example, `for (; i<1; i=i+1)` should produce a parse error because the first expression is missing. For the AST implementation, create a class ForStmt similar to WhileStmt and IfStmt.

Generalize binary operators. Modify Scanner.jflex and Parser.cup to also handle binary operators <, <=, >, >=, ==, and != (Sec 6.5.8 and 6.5.9). For simplicity, we will **not** include in simpleC the logical AND/OR binary operators (Sec 6.5.13 and 6.5.14). Make sure that precedence and associativity for all 17 operators (11 from Project 1 and 6 from Project 2) are specified correctly in Parser.cup. Add parser actions to generate the corresponding AST nodes for the 6 new operators. Modify ast/BinaryExpr.java to handle these new operators.

Conditionals. In C, **any** expression of scalar type can be used as a conditional in if-then, etc. Since in simpleC each expression is of scalar type (either int or double), any expression can be the conditional in if-then, if-then-else, while-loop, and for-loop. For example, `if(x)…` and `while(x-=3)…` are allowed.

No semantic checks. As with Project 1, do not worry about semantic checks.

Testing

Your submission must work correctly on test program lpc.c provided on the web page. *You can expect that a substantial number of points in the grading will be related to this test case.* Make sure that your project can produce a compilable and executable C program:

```
./simplec lpc.c > mylpc.c
gcc -o lpc main.c mylpc.c
./lpc
```

You can use the following main.c as a test driver:

```
#include <stdio.h>
double lpc(void);
int main(void) { double res = lpc(); printf("%lf",res); }
```

Use this main to run the original lpc.c. Ensure that the printed value is the same as the one printed when using mylpc.c.

In addition to this test case, 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 p2
make clean
cd ..
tar -cvzf p2.tar.gz p2
```

Then submit `p2.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.