Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu), Classes: MW 4:40-6:30pm, DLC304, Office Hours: MW 2-3pm.

Project 2: MINI Parser
(Due Monday 10/24/11)

This project is to implement a first version of a parser for the MINI programming language. In this version, no syntax tree is generated; the parser either accepts the input program as correct (with no output), or reports a syntax error it finds in the program. The implementation is to be done with JavaCC.

You need to take two steps to do this project. First, you need to transform the raw grammar given in the MINI language manual into an LL grammar; second, you use JavaCC to implement a topdown parser using the LL grammar.

Transforming the Grammar

The official MINI grammar is given in the *MINI Programming Language Manual (version 1.6 Fall '11)*. However, the given grammar is not directly suitable for topdown parsing. It is ambiguous, and it contains left-recursions and common prefixes. You need to use the techniques discussed in class to transform the grammar to an equivalent LL(1) or LL(2) form. Specifically, you need to:

- Use the operator precedence and associativity information to rewrite the expression section of the grammar to eliminate the well-known ambiguity in binary expressions.
- Eliminate all left-recursions in the grammar using the standard technique. The main occurrences of left-recursions are also in the expression section.
- Use the "left-factoring" technique to reduce the need for multiple-token lookahead as much as possible. It may be difficult to factor out all the occurrences of common prefixes and produce an LL(1) grammar, so for this project, we also accept LL(2) form.

You should follow the above steps in the order given. If you perform the steps in a different order, the task could be substentially harder.

Note that you don't have to deal with the "dangling else" ambiguity problem. You may leave the grammar for `If` statement as is, since JavaCC has the default resolution of matching an *else* with the innermost *if* statement.

Save a copy of your LL grammar in a text file `miniLLG.txt`, and later include it as part of your submission. (*Don't forget to include your name in the file.*)

Implementation

Download the tar file, `proj2-code.tar`. Decompress it to setup the directories. You should call your parser program `miniParser0.jj`, and place it in the subdirectory `parser0`. Don't forget to include a line `package parser0;` at the top of the program, between the `PARSER_BEGIN` and `PARSER_END` brackets.

You need to copy and paste the token specification from your `miniLexer.jj` to the top section of this parser file. There is no need to copy the file `miniLexer.jj` itself to proj2 directory.

The mapping from the LL grammar to JavaCC specification should be straightforward. Do not change JavaCC's default lookahead setting (which is "single-token"). You should only use two-token lookahead occasionally, by inserting the directive `LOOKAHEAD(2)` at places as needed.

Add a comment block in front of each parsing routine to show the corresponding grammar rule, *e.g.*

```
// Program -> ClassDecl {ClassDecl}
//
void Program():
{}
{ ... }
```

Running and Testing

A driver program (`parser0/TestParser0.java`), a `Makefile`, a shell script (`runp0`), and a set of test programs (in `tst`), are provided to you for building and testing your parser.

On a syntactically-correct program, your parser should run silently, and the driver will print out a message, "Parsing successful", at the end. If the input program contains syntax errors, you parser should report the nature and location of the first error.

What to Turn in

Submit your LL grammar file `miniLLG.txt`, and your program file `miniParser0.jj` through the "Dropbox" on the D2L class webpage.