CS 164, Spring 2005 **CS 164: Homework #3** **P. N. Hilfinger**

Due: Friday, 18 February 2005

General instructions about homework. Use the command '`submit hw3`' to submit your homework. We will not accept homework in any other form. Unless the problem specifies otherwise, please put your solutions in a file named `hw3.txt`.

1. Consider the following ambiguous grammar:

```
prog  →  ε
prog  →  expr ';'
expr  →  ID
expr  →  expr '-' expr
expr  →  expr '/' expr
expr  →  expr '?'  expr ':'  expr
expr  →  '(' expr ')'
```

The start symbol is `prog`; `ID` and the quoted characters are the terminals. An `ID` is a single letter. Assume the same precedence and association rules as in C (or C++ or Java). Write a recursive-descent parser for the language described that converts these expressions to Lisp notation and then prints them: `x-y` becomes `(- x y)`; `x?y:z` becomes `(if x y z)`, etc. Put your program in a file called P1.java (see the template in `~cs164/hw/hw3/P1.java`). Your program need not create a parse tree nor an AST; it just needs to print.

2. Consider the original, ambiguous grammar in problem 1, above.

a. Produce an (improper) LL(1) parsing table for this grammar. Since it is ambiguous, some slots will have more than one production; list all of them. Show the FIRST and FOLLOW sets.

b. Modify the grammar to be LL(1) and repeat part a with it.

In this case, we're just interested in recognizing the language, not in printing Lisp expressions, so don't worry about preserving precedence.

3. [From Aho, Sethi, Ullman] A grammar is called ϵ-*free* if there are either no ϵ productions, or exactly one ϵ production of the form $S \rightarrow \epsilon$, where S is the start symbol of the grammar, and does not appear on the right side of any productions. (We write ϵ productions either as '$A \rightarrow$ ' or '$A \rightarrow \epsilon$'; both mean the same thing: there are no terminals or non-terminals to the right of the arrow). Describe an algorithm to change a grammar into an equivalent ϵ-free grammar (i.e., one recognizing the same language). By "describe," I mean "give sufficient detail that a programmer could probably figure out what you meant and convert it into a program." Apply your algorithm to the grammar:

```
S  →  aSbS | bSaS | ε
```