Computer Science 131, Spring 2002
Assignment 2: Syntax

Out: Wednesday February 6

Due: Monday, February 11, in class

This assignment requires no programming, just written responses. To help the graders, your answers should be typed up rather than handwritten.

1. In class you were given a copy of the grammar for the concrete syntax of the C language, taken from K&R. Is this grammar LL(1)? Explain why or why not.

2. Consider the following grammar for regular expressions:

$$\begin{array}{rl} R \rightarrow & \epsilon \\ | & \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \ldots \\ | & R\,R \\ | & R + R \\ | & R^* \end{array}$$

   As shown, derivations in the this grammar might be a reasonable representation of an abstract syntax for regular expressions (i.e., trees in this grammar can be used to represent regular expressions). However, this grammar is less satisfactory as a definition for a concrete syntax of regular expressions.

   (a) Show that the above grammar is ambiguous by providing a string which can be parsed in more than one way.

   (b) Give an unambiguous grammar for a concrete syntax of regular expressions, subject to the following constraints:

   - The concrete syntax should permit parenthesized regular expressions
   - Star should bind more tightly than concatenation, which should bind more tightly than +.
   - Concatenation should be left-associative, but + should be right-associative. (For regular expressions the choice of associativity doesn't affect the meaning at all, but we need to make *some* choice in order for the grammar to be unambiguous.)

   Your grammar need not be LL(1).

3. (a) Consider the grammar

$$\begin{array}{lll} S \rightarrow & E\ \$ & \\ E \rightarrow & T & \\ \quad | & E + T & \\ T \rightarrow & n & \text{(where } n \text{ represents any integer constant)} \end{array}$$

This grammar is not LL(1). Is it LL(k) for any integer k? Carefully justify your answer.

(b) Consider the grammar

$$\begin{array}{lll} S \rightarrow & E\ \$ & \\ E \rightarrow & T & \\ \quad | & E + T & \\ \quad | & E - T & \\ T \rightarrow & n & \text{(where } n \text{ represents any integer constant)} \end{array}$$

This grammar is also not LL(1). Is it LL(k) for any integer k? Carefully justify your answer.

4. Several designers of the programming language Dylan came from the Lisp community, and thus Dylan started out life with a concrete syntax very similar to that of Lisp and Scheme (using S-expressions and prefix notation), e.g.,

```
(define-method double (x)
   (* 2 x))
```

Later an alternate *concrete* "infix" syntax was developed, closer to that of most imperative programming languages:

```
define method double(x)
  2 * x;
end method;
```

The plan was that the language would have a single abstract syntax, but two different concrete syntaxes. Programmers could choose whether to write files of code in prefix-style or infix-style; the compiler wouldn't care, since source programs in either syntax could be mapped the same set of internal abstract syntax trees. If desired, automated tools could easily convert code from one syntax to the another.

Should programming languages support more than one concrete syntax? Answer this question in three paragraphs. In the first paragraph, argue as strongly as possible why one might want to have a language with multiple concrete syntaxes; in the second paragraph, argue as strongly as possible why it might be a bad idea. Finally, explain which side you personally find most convincing.