Compilers Comprehensive Exam
Fall 2004

This is a 60 minute, closed book exam. Please mark your answers in the blue book.

1. **Regular Expressions** (5 points)

   Consider a language where real constants are defined as follows: A real constant contains a decimal point or E notation, or both. For instance, 0.01, 2.71821828, ~1.2E12, and 7E~5 are real constants. The symbol "~" denotes unary minus and may appear before the number or on the exponent. There is *no* unary "+" operation. There must be at least one digit to left of the decimal point, but there might be no digits to the right of the decimal point. The exponent following the "E" is a (possibly negative) integer.

   Write a regular expression for such real constants. Use the standard regular expression notation described by the following grammar:

   $$R \to \epsilon \,|\, \text{char} \,|\, R + R \,|\, R* \,|\, RR \,|\, (R)$$

   You may define names for regular expressions you want to use in more than one place (e.g., *foo* = R).

$$\begin{aligned}
\mathtt{digit} &= \mathtt{0+1+2+3+4+5+6+7+8+9} \\
\mathtt{posint} &= \mathtt{digit\ digit}^{*} \\
\mathtt{int} &= (\epsilon + {}^{\sim})\ \mathtt{posint} \\
\mathtt{exp} &= \mathtt{E\ int} \\
\mathtt{frac} &= \mathtt{.\,digit}^{*} \\
\mathtt{real} &= (\mathtt{int\ frac}\ (\mathtt{exp} + \epsilon)) + (\mathtt{int}\ (\mathtt{frac} + \epsilon)\ \mathtt{exp})
\end{aligned}$$

2. **Grammars** (20 points)

Consider the following grammar. The nonterminals are E, T, and L. The terminals are +,**id**,(,), and ;. The start symbol is E.

$$\begin{array}{rcl} \mathrm{E} & \rightarrow & \mathrm{E} + \mathrm{T} \,|\, \mathrm{T} \\ \mathrm{T} & \rightarrow & \mathbf{id} \,|\, \mathbf{id}() \,|\, \mathbf{id}(\mathrm{L}) \\ \mathrm{L} & \rightarrow & \mathrm{E}; \mathrm{L} \,|\, \mathrm{E} \end{array}$$

Give an LL(1) grammar that generates the same language as this grammar. For full credit, you must show (convincingly) that your grammar is LL(1).

(a) Eliminate left recursion:

$$\begin{array}{rcl} \mathtt{E} & \rightarrow & \mathtt{TE}' \\ \mathtt{E}' & \rightarrow & +\mathtt{TE}' \,|\, \epsilon \\ \mathtt{T} & \rightarrow & \mathtt{id} \,|\, \mathtt{id}() \,|\, \mathtt{id}(\mathtt{L}) \\ \mathtt{L} & \rightarrow & \mathtt{E}; \mathtt{L} \,|\, \mathtt{E} \end{array}$$

(b) Left factor:

$$\begin{array}{rcl} \mathtt{E} & \rightarrow & \mathtt{TE}' \\ \mathtt{E}' & \rightarrow & +\mathtt{TE}' \,|\, \epsilon \\ \mathtt{T} & \rightarrow & \mathtt{id}\,\mathtt{T}' \\ \mathtt{T}' & \rightarrow & \epsilon \,|\, (\mathtt{T}'' \\ \mathtt{T}'' & \rightarrow &) \,|\, \mathtt{L}) \\ \mathtt{L} & \rightarrow & \mathtt{EL}' \\ \mathtt{L}' & \rightarrow & ;\mathtt{L} \,|\, \epsilon \end{array}$$

(c) Check that it's LL(1). For this part you just needed to give enough information to show that there would be no conflicts in the parsing table. The following is sufficient:

$$
\begin{array}{lclrcl}
\mathrm{E}' & \rightarrow & +\mathrm{TE}' \,|\, \epsilon & \mathtt{First}(+\mathrm{TE}') & = & \{+\} \\
 & & & \mathtt{Follow}(\mathrm{E}') & = & \{\$,), ;\} \\
\mathrm{T}' & \rightarrow & \epsilon \,|\, (\mathrm{T}'' & \mathtt{First}((\mathrm{T}'') & = & \{(\} \\
 & & & \mathtt{Follow}(\mathrm{T}') & = & \{+, \$,), ;\} \\
\mathrm{T}'' & \rightarrow &) \,|\, \mathrm{L}) & \mathtt{First}()) & = & \{)\} \\
 & & & \mathtt{First}(\mathrm{L})) & = & \{\mathrm{id}\} \\
\mathrm{L}' & \rightarrow & ;\mathrm{L} \,|\, \epsilon & \mathtt{First}(;\mathrm{L}) & = & \{;\} \\
 & & & \mathtt{Follow}(\mathrm{L}') & = & \{)\}
\end{array}
$$

3. **Parsing** (15 points)

Consider the following grammar. The nonterminals are S' and S. The terminals are **op** and **x**. The start symbol is S'.

$$\begin{aligned} S' &\rightarrow \mathrm{S} \\ \mathrm{S} &\rightarrow \mathrm{S}\ \mathbf{op}\ \mathrm{S} \mid \mathbf{x} \end{aligned}$$

(a) Draw the DFA built from sets of LR(0) items for this grammar. Show the contents of each state. (Note: Don't augment the grammar with a new start symbol.)

```
                  -----------------
                  | S' -> . S     |
                  | S -> . S op S |
                  | S -> . x      |
                  -----------------
           S         |           |      x
      ----------------          -------------
      V                                     V
-----------------                    -------------
| S' -> S .     | **                 | S -> x . |
| S -> S . op S |                    -------------
-----------------                           ^
      |            op            x          |
      |----------------      ----------------
                      V       |
                  -----------------
                  | S -> S op . S |
                  | S -> . S op S |
                  | S -> . x      |
                  -----------------
```

(b) Is this grammar LR(1)? Briefly explain why or why not.

```
No.  The grammar is ambiguous.
```

4. **Scope** (10 points)

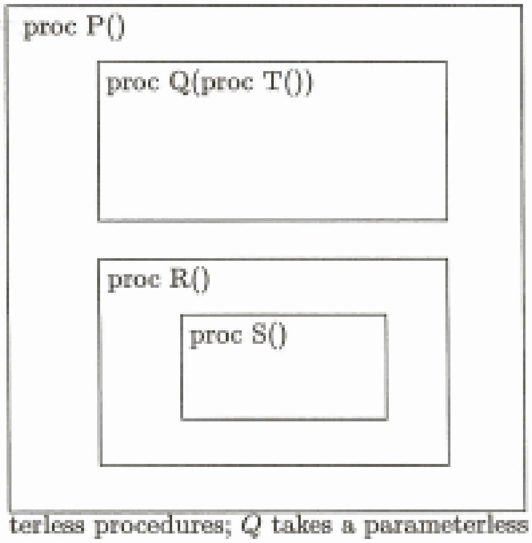
Give a simple program that produces different results if executed using lexical scoping than if executed using dynamic scoping. Show what your program produces in both cases. Use any reasonable and clear programming notation.

```
main()
    var x;

    proc p1
        var x;
        x := 1;
        p2();
    end;

    proc p2
        print(x);
    end;

x := 0;
p1();
end;
```

`This program prints 0 with lexical scope and 1 with dynamic scope.`

5. **Activation Records** (10 points)

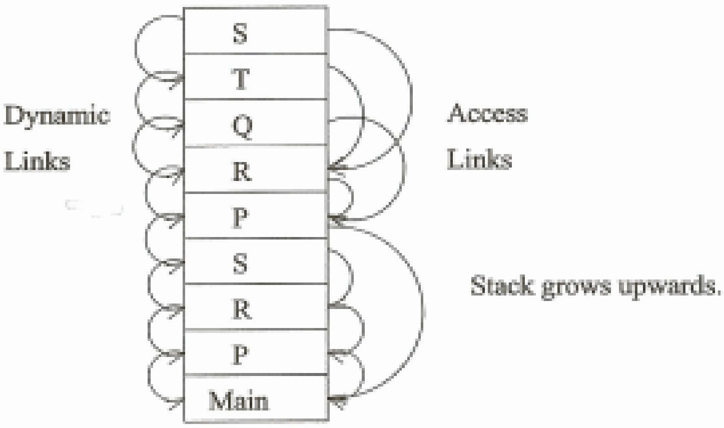
Consider a program with the following lexical structure. The program is written in a lexically scoped language with nested procedures (like Pascal):



P, R, and S are parameterless procedures; Q takes a parameterless procedure T as a parameter. Suppose that at run-time the following sequence of calls is made:

P is called from some lexically-enclosing main program
P calls R
R calls S
S calls P
P calls R
R calls Q with S as a parameter
Q calls T
T calls S

Draw the stack of activation records present after this sequence of calls. You don't need to show the entire contents of the activation record—for each indicate only the name of the procedure being activated, the control (dynamic) link for that activation record, and the access (static) link for that activation record.



Stack grows upwards.