CS4905 Introduction to Compiler Construction
Assignment 3 Feb. 8, 2007 due Feb. 20, 2007
This assignment is worth 5% of your final mark. References to exercises are in the course text. Assignments must be your own work.

1. Textbook exercise 3.10, p.84 on identifying LR(1) states and constructing an LR(1) parse table.

2. Textbook exercise 3.13, p.84 on showing why a grammar is LALR(1) but not SLR.

3. Textbook exercise 3.14, p.85 on showing why a grammar is LL(1) but not LALR(1).

4. Textbook exercise 3.15, p.85 on overcoming LALR(1) conflicts with yacc precedence directives.

5. Write a yacc program that will take a series of Roman numerals as input, parse them, and take semantic actions to compute and print the base 10 number corresponding to the Roman numerals. Remember that the roman numerals are I(=1), V(=5), X(=10), L(=50), C(=100), D(=500) and M(=1000). The rules for roman numerals are

   (i) If a letter is immediately followed by one of equal or lesser value, the two values are added (e.g. XV = 15, MCXX = 1120).

   (ii) If a letter is immediately followed by one of greater value, the first is subtracted from the second (e.g. IV = 4, XL = 40, CM = 900). For this rule, the first letter must represent a power of ten, and the second letter can be at most 10 times the first letter. This means that I can precede V or X, X can precede L or C, and C can precede D or M. This follows the "conventional principle" of Roman numerals.

   Other examples are XLVII = 47, MCMXIV = 1914. Assume a maximum number of MMM. For example, the Roman numeral MCMXIV, when parsed and interpreted should result in the base 10 number 1914. Test your program on the following Roman numerals:

```
CDIX
MMD
MMCMXCIX
CMCCLXIV
MMDCCLII
MMVII
MMDCXL
MCMXXXVII
MDCLVI
XXXIX
XCIX
CMXCIX
MMXII
CXLVII
```

Hints: Use the desk calculator example from Lab 2 as a starting point, and change the yacc stack type from `double` to `int`. One possible grammar for Roman numerals is as follows:

<N> ::= G | G <N>

where G is a token (one of the seven possible letter “glyphs”) returned from `lex`. Remember that `yylval` contains the returned value of a `lex` token.

6. Textbook exercise on writing an interpreter using JavaCC, 4.3 on p.102. Note that this first requires you to modify the grammar 3.15 to accommodate the negative integer specification and the "`print`" statement. Left recursion removal is also required. You will also need a symbol table that can store the values of integer variables. For this, you can modify the hash table given in Program 5.2 (p.106) to store the integer value as the variable's binding (as in Lab 3). Test that your interpreter works using the following input file:

```
k = -2;
n = 6;
j = k - n * 3 + n / k;
r = j - ( 2 / k ) * n;
i = r / (j + 2 ) - k;
print j;
print r;
print i;
p = 27 - i * 12 / ( r - j);
n = (n + r) / k;
print p;
print n
```

Your interpreter should print the values of `j, r, i, p` and `n` correctly.

Remember that all assignments must be handed in on 8.5 by 11 inch paper, typed, with a table of contents and all pages numbered. For questions 5 and 6, make sure you hand in a complete listing of your input program, along with a copy of the output resulting from the testing you did with the input test data files.