Programming Language Concepts, CS2104
Tutorial 3 (16 September 2011)
(All students must prepare/attempt in advance.)

Exercise 1

Define a Prolog predicate that takes two list arguments, and succeeds when the second list is the reverse of the first.

Exercise 2

Write a Prolog program that implements the *quicksort* algorithm. Consider using the cut (!) to make your program more efficient.

.

Exercise 3

A type of virtual machine that is often used as the target architecture of compilers is the *stack machine*. A simplified version of such a machine has the following features:

- A *stack* that can store integers, which can be accessed in a last-in-first-out fashion.
- Two types of instructions:
  - Data transfer instructions: `push C`, where `C` is a constant; this instruction pushes the constant `C` onto the top of the stack.
  - Arithmetic instructions: `add`, `sub`, `mul`, `div`; these instructions pop two integers from the stack, perform the operation indicated by the instruction's name, and push the result back.
- A `program` is a sequence of instructions, separated by semicolon; the instructions are executed in the order in which they appear in the program.
- At the end of the program's execution, the result of the program is the integer found at the top of the stack.

Write a Prolog predicate that simulates an execution of such a machine. Your predicate should take in as argument the program to be run, and should return the result of the program.

Sample run of the program:

```
?- exec((push 2 ; push 3 ; push 4 ;  add ; mul), Result).
Result = 14
```

The program pushes the values `2`, `3`, and `4` on the stack. Then, the `add` instruction pops `3` and `4`, and pushes back `7`. Further, the `mul` instruction pops `7` and `2`, and pushes back `14`. Since the program has terminated, the result is the integer at the top of the stack, which is `14`.

Exercise 4

Regular arithmetic expressions can be compiled into the "assembly" language of the stack machine described in the previous exercise. For simplicity, we shall only consider expressions having the binary operators `+`, `-`, `*`, and `/`. For example, the expression:

```
(2+3)*(4+5-3)+10/2
```

Can be translated into:

```
push 2 ;
push 3 ;
add ;
push 4 ;
push 5 ;
add ;
push 3 ;
sub ;
mul ;
push 10 ;
push 2 ;
div ;
add
```

The execution of the program produces a result that is equal to the value obtained by evaluating the original expression using the predicate `is`. Write a Prolog program that performs this translation.

Hint: due to the associativity of the Prolog ';' (semicolon) operator, the programs that you produce may appear "entangled". You will need to write a helper predicate to syntactically dis-entangle the program.