CS2104 Programming Language Concepts
Laboratory Assignment 3 : An Arithmetic Calculator

(Due : 8th October 2013 8pm)

Please submit your solution into IVLE workbin as a single OCaml file. You can use the utilities provided by globals.ml, debug.ml, gen.ml, etc but must not change their implementation. These utility modules need not be submitted.

You have been given a partial lexical analyser and a parser for a simple arithmetic expression that can support the following grammar form:

```
<expr> ::= <number> | <identifier> | <expr> op <expr> | ( <expr> )
<op> ::= + | - | * | /
```

Your task is to extend the arithmetic expression form to support a negation operation (denoted by ~ <expr>) and a let construct that can be used to bind values to identifiers. The new grammar form is expected to be:

```
<expr> ::= <number> | <identifier> | <expr> op <expr> | ( <expr> )
           | ~ <expr> | let <identifier> = <expr> in <expr> |
<op> ::= + | - | * | /
```

We have already provided an abstract syntax for your arithmetic expression calculator:

```
type exp = ENum of Num.num
        | EId of string
        | EPlus of exp * exp
        | EMinus of exp * exp
        | EDiv of exp * exp
        | ETimes of exp * exp
        | ELet of string * exp * exp
```

The negation operation can be implemented using subtraction. Complete the following tasks for your calculator:

(i) Change the scanner to support the recognition of integer by completing the method Lexical.numeric

(ii) Change the parser to support let-binding and negation operator by adding the new parser terms to Calc.factor

(iii) Complete the implementation of the arithmetic evaluator, Calc.eval

We have provided some tests for you whose expected solutions are in expected-lab3.txt. You are advised to write more test cases yourself.

Bonus Section (10%)

This is meant for those who like to practice more. In this section, you are to extend your arithmetic expression to the following grammar form:

```
<expr> ::= <number> | <identifier> | <expr> op <expr> | ( <expr> )
             | ~ <expr> | let <identifier> = <expr> in <expr>
             | <expr>! | <expr>++
<op> ::= + | - | * | / | ^
```

This new grammar has the following new features:

(i) A power operator ^ that would evaluate 3^2 to 9. The precedence of ^ is lower than +, and it is right-associative. For example 3^2^3 is parsed as 3^(2^3). You may throw an exception if the second input is negative.
(ii) A post-fix operator ! for computing factorial. That is 3! returns 3*2*1 = 6. You may throw an exception if the input is negative.
(iii) A post-fix operator ++ to capture increment by one. That is 5++ would return 6.

Extend your scanner and parser to support the extended calculator. Refine the evaluator (or interpreter) to support the new operations. In particular, you would need to change Lexical.symbolic so that it recognizes “++” as a single operator instead of two operators “+” followed by “+”.