COMP 302: Assignment 3, Summer 2008.
SML datatypes, closures

Due Date: June 27th in my office (McConnell 106) or on June 26th in class.

Guidelines for submission: For this assignment, please print out a few test cases to demonstrate the correctness of your work. I emphasize that *hand-written outputs are not sufficient.*

Question 1: (30 points) Recall the logical expressions defined in the last question of the midterm:

- $true$ and $false$ are logical expressions,
- variables x, y, and z are logical expressions,
- if ϕ is a logical expression, then $\neg\phi$ is a logical expression,
- if ϕ_1 and ϕ_2 are logical expressions, then $\phi_1 \wedge \phi_2$ (the *and* operator) and $\phi_1 \vee \phi_2$ (the *or* operator) are logical expressions.
- if ϕ is a logical expression and v is a variable, then $\exists v \phi$ is a logical expression.

A variable is *bound* if it is enclosed within an $\exists$ quantifier. Otherwise it is *free.* If an expression contains no free variable we say it is *closed.* We will use the following datatype to encode logical expressions:

```
datatype var = X | Y | Z;

datatype LExp = True | False | Var of var |
            Not of LExp | And of LExp * LExp |
            Or of LExp * LExp | Exists of var * LExp;
```

We can define the value of a closed expressions inductively as follows:

- the value of $true$ and $false$ are $true$ and $false$ respectively,
- if ϕ has value $true$, then $\neg\phi$ has value $false$ and vice-versa,
- $\phi_1 \wedge \phi_2$ has value $true$ if both ϕ_1 and ϕ_2 are true, and false otherwise, $\phi_1 \vee \phi_2$ has value $true$ if either ϕ_1 or ϕ_2 are true and $false$ otherwise.

- $\exists v \phi$ is true if there exists an assignment $a \in \{true, false\}$ to the free occurrences of v in ϕ which makes ϕ evaluate to true.

Now, construct a function of type `LExp-> bool option` which returns `NONE` if the expression is not closed, otherwise it returns `SOME(a)` where `a` is the value of the input expression.

Question 2: (30 points) Suppose we have a numbering system with multiplication and addition defined. We can encode this into a structure matching the following signature:

```
signature RING =
   sig
      type t
      val add: t*t->t
      val mult: t*t->t
      val ONE: t
      val ZERO: t
      val toString: t->string
   end;
```

Implement structures matching the signature above for a) ints, b) reals, c) rationals, and d) bignums.

Question 3: (40 points) The following is a generic template for expression trees consisting of addition and multiplication operators:

```
datatype 'a expr = Add of 'a expr * 'a expr |
         Mul of 'a expr* 'a expr | Leaf of 'a

signature EXP =
  sig
    structure num: RING
    val eval : (num.t expr)->(num.t)
    val sop : (num.t expr)->(num.t expr)
  end;
```

Write a functor which takes as input a structure `R` matching the `RING` signature and returns a structure matching the signature `EXP where num.t = R.t`. The function `eval` should take an expression tree (i.e. a value of type `num.t expr` as input and apply the appropriate operations to evaluate the tree. The function `sop` should take as input an expression tree and rearrange the operators so that no addition operations occur below multiplication operators. You can (and need to) assume that multiplication distributes over addition.