University of Calgary
Department of Computer Science

Introduction to Programming Paradigms (CPSC 449)
Assignment 3 of 4

This assignment is to be completed individually and you will only receive credit for your own original work.

Your submission will be evaluated with respect to style and efficiency, as well as effectiveness.

Reproducing or adapting any published or unpublished material, regardless of the source, without including a proper reference to the source is considered academic misconduct and may result in very severe penalties.

1. Develop a pseudorandom number generating function that is a given a seed value as an argument and will produce a value between 0 and 10. You will need to investigate how this can be done, but as a clarifying example consider the pseudorandom sequence 5, 13, 17, 19, 20, 10 ..., where the first element of the sequence is 5 and the i+1th element of the sequence is the result of the expression mod (i*11) 21. (n.b., your pseudorandom number generating function is expected to be superior than the one detailed above.)

Just like arithmetic expressions, expressions in propositions logic can be represented using trees. The Expression and Operator types defined below are sufficient for representing logical expression using only the conjunction (AND) and disjunction (OR) operations.

```
data Expression = Literal Bool | Operation Operator Expression Expression
data Operator = AND | OR
```

2. Write a height function `height` that will count the number of edges (in the tree representation) on the longest path from the root to any leaf node (i.e., Literals). Please note that the height of a tree that is just a single Literal should be reported as 0.

3. Write an evaluation function `eval` that takes an Expression as an argument and produces a Bool return value corresponding to the result if the Expression were to be evaluated. As a clarifying example, the function call:

```
> eval (Operation OR (Literal True) (Operation AND (Literal True) (Literal False)))
```

would be expected to result in the return value `True`.

4. Alter the type definitions so that you can also represent and evaluate negation operations (i.e., logical not) and exclusive disjunction operations (i.e., logical xor).

5. Write a function to show the expression, using the characters T and F instead of True and False, respectively, and using the caret symbol ^ for conjunction, the lowercase `v` for inclusive disjunction, the plus sign + for exclusive disjunction, and the tilde ~ for negation. Your expression should be constructed with bracketing around each operation, but bonus marks will be rewarded if you can avoid wrapping unnecessary brackets around literals. To clarify, a function call to show this expression:

```
(Operation OR (Literal True) (Operation AND (Literal True) (Literal False)))
```

would be accepted with full marks if it returned the string "`(T) v ((T) ^ (F))`"

and it would be accepted with a small bonus if it returned the string "`T v (T ^ F)`"