Homework 3

Please submit this work by Wednesday, April 23 at 11:59pm.

Problem 1: The Programming Language L2

Terms in the language L2 are as follows:

```
t ::= n           // nonnegative integer constants n, one or more digits
    | T           // true
    | F           // false
    | x           // variable names, one or more lowercase letters
    | ~(t)        // logical negation (tilde character)
    | Z(t)        // "is zero" test
    | <(t,t)      // less than
    | +(t,t)      // addition
    | LET(x<-t,t) // binding: "let x be t in t"
    | IF(t,t,t)   // conditional
```

This is an enrichment of last week's language L with boolean constants, some new operations and a new IF form for conditional evaluation. As in L, whitespace is allowed between L2 tokens and the lexical scanner must tolerate it.

Your task this week is to implement L2, and furthermore to implement an optimization that removes unused variables as a separate pass during compilation.

Construct your L2 implementation by copying your L source code into a new directory `hw3/L2`. (Be careful not to copy *everything* from `hw2/L` into this directory, because you don't want to copy all the hidden `svn` files.)

Implementation of L2 will consist largely of making straightforward modifications to your implementation of L. You will need new tokens, an enhanced lexer and an enhanced parser, new AST forms, new ScopedAST forms. Furthermore, since evaluation of an L2 program can yield an integer or a boolean, your evaluator needs a new type. Whereas in L the function `Eval.eval` had type`ScopedAST.exp -> value` where `value` was a synonym for `int`, in L2 the same function has the type `ScopedAST.exp -> Value.value` where the module `Value` is as follows:

```
structure Value = struct
  datatype value
    = Int of int
    | True
    | False
  fun tos (Int n) = Int.toString n
    | tos True = "T"
    | tos False ="F"
  fun eq (t:value,u) = (t=u)
end
```

Furthermore, unlike L programs, evaluation of scannable, parseable, scopeable L2 programs is no longer guaranteed to proceed without error, because there exists the possibility of nonsensical programs like `+(8,T)` or `IF(12,13,14)`. The complete set of nonsensical L2 programs is as follows:

- logical negation of a number,

- testing whether a boolean constant is zero,
- performing a less-than comparison where one or both arguments are booleans,
- "adding" a boolean, or
- having a number in the test position in a conditional.

Each of these constitutes a new kind of error, a*runtime error*. Therefore we add `RuntimeError` to the errors we associated with the previous phases:`SyntaxError`, `ParseError` and `UnboundVariable`. (It is worth making clear that we do not loosely equate numbers and booleans as C does.)

For these reasons, the `EVAL` signature for L2 needs to look like this:

```
signature EVAL = sig
  exception RuntimeError of string
  val eval : ScopedAST.exp -> Value.value
end
```

Please do not impose a type discipline on L2; the language specifically lacks a static type system. Raise`RuntimeErrors` as they arise during evaluation.

Besides implementing L2 with its enriched set of values and operations, implement a useless-variable-elimination optimization on scoped ASTs that adheres to the following signature:

```
signature OPTIMIZE = sig
  val optimize : ScopedAST.exp -> ScopedAST.exp
end
```

The optimization should transform programs such that any variable that is bound but not used should be eliminated. For example, the program`"LET(x<-1,0)"` should be transformed simply to `"0"`, and `"LET(x<-1,LET(y<-2,+(x,1)))"` to`"LET(x<-1,+(x,1))"`. (Of course, all useless variables can be "eliminated" simply by evaluating the program, but here we are providing experience with program transformation and compiler structure.)

As such, the `Interpret` structure can be rewritten as follows:

```
structure Interpret : INTERPRET = struct
  val interpret = Eval.eval o
                  Optimize.optimize o
                  Scope.scope o
                  Parse.parse o
                  Lex.lex
end
```

In summary, problem 1 consists of doing the following:

- copying your `hw2/L` source code (but not the hidden svn baggage) into a new directory`hw3/L2`,
- including the `Value` structure defined above,

- modifying the `EVAL` signature as above,
- hacking on the lexer, parser, scoper and evaluator to support L2, raising `RuntimeErrors`for programs that are ultimately nonsensical but otherwise fine (with respect to the previous compilation phases),
- adding the `OPTIMIZE` signature as above and implementing an `Optimize` structure to match it,
- modifying `Interpret` to include the optimization pass.

You are encouraged to fix and/or improve your L source code as part of your work on L2.

Problem 2

TaPL p. 24 gives a grammar for a simple term language, not altogether unlike L2. As presented in lecture last week, the grammar generates terms that can't be evaluated, like `iszero false` (an unevaluatable term). In a programming language implementation, the impasses in evaluating terms like `iszero false` correspond to runtime errors.

Rewrite the grammar on p. 24 such that it produces only terms that will never lead to a nonsensical state (runtime error) at any point in their evaluation, even taking into consideration that for every conditional, either branch may be evaluated. For example, the grammar should generate `succ 0` but not `succ false`, and all terms `if true then ...` but no terms`if 0 then ...`. We expect only a simple plaintext presentation.