CSE 3302 Lab Assignment 2

Due October 5, 2012

Goal:

Understanding of elementary parsing, infix-to-postfix conversion, and type checking.

Requirements:

1. Design, code, and test a program to parse and type check an expression with the lexical elements of lab 1. The input is:

   a. A line with three non-negative integers `m`, `n`, and `p`.
   b. `m` lines, each with the *name* of a boolean identifier.
   c. `n` lines, each with the *name* of an integer identifier. (All of these will be different from the boolean identifiers.)
   d. `p` lines corresponding to the output format of lab 1 (token and classification).

   The output is an abstract syntax tree similar to that produced by `infix2postfix.ast.c`. Each node should also indicate the associated type. If a type inconsistency is detected, your program should terminate with an error message. There are several strategies for detecting type errors.

2. Email your program to `mehra.nourozborazjany@mavs.uta.edu` by 12:45 p.m. on October 5.

Getting Started:

1. You may use any language you wish, but please check with the TA or me if you plan to use anything "exotic". If you use JavaScript, then you may use an input box for pasting the input file along with an output box.

2. You are allowed to use available code. Be sure to provide appropriate attribution.

3. Even though the language is "`C` like", booleans as `T` and `F` (like Java) does not allow the usual flexibility of an expression like:

   `(a==1)+(b<10)`, but this could be coded as `(a==1 ? 1 : 1-1) + (b<10 ? 1 : 1-1)` or as
   `a==1 ? (b<10 ? 2 : 1) : (b<10 ? 1 : 1-1)` or even as
   `a==1 ? b<10 ? 2 : 1 : b<10 ? 1 : 1-1`

4. Unlike other operators, whose result and operand types are fixed:

   test `?` *trueExp* `:` *falseExp* needs to have *trueExp* and *falseExp* with the same type that also determines the result type. (*test* must be boolean)

   The *condop*s (<, >, ==, !=, <=, or >=) may have operands that are boolean or numeric, but both operands must have the same type (so with booleans, == is like *iff* and <= is like *implies*, e.g. `F<T` evaluates to `T`).

5. `C` precedence rules apply (so do the ones for Java . . .).

6. All identifiers used as tokens will appear in the boolean identifier list or the integer identifier list. Operators and their classifications will be correctly constructed.

7. If a parse error is detected, you may stop immediately.

```
$(a+b)*(c+d)<e+f*g|h*i+j<k+l*m*n#
Postfix: ab+cd+*efg*+<hi*j+klm*n*+<|
```

```
AST:
        n (31,-1,-1)
      * (30,28,31)
          m (29,-1,-1)
        * (28,27,29)
          l (27,-1,-1)
    + (26,25,30)
      k (25,-1,-1)
  < (24,22,26)
      j (23,-1,-1)
    + (22,20,23)
        i (21,-1,-1)
      * (20,19,21)
        h (19,-1,-1)
| (18,12,24)
        g (17,-1,-1)
      * (16,15,17)
        f (15,-1,-1)
    + (14,13,16)
      e (13,-1,-1)
  < (12,6,14)
        d (10,-1,-1)
      + (9,8,10)
        c (8,-1,-1)
    * (6,3,9)
        b (4,-1,-1)
      + (3,2,4)
        a (2,-1,-1)
```