Exercise Compiler Construction (7)

Hans de Nivelle

Due: December 9th 2009

1. Consider the class `token` that was used in the calculator of the last exercise. (it can be downloaded with **maphoon2008.tar.gz**.) We restrict our attention to the subset of tokens that is defined by[1]

   ```
   tkn_IDENTIFIER, tkn_NUMBER, tkn_PLUS, tkn_TIMES,
   tkn_MINUS, tkn_DIVIDES, tkn_FACULTY
   ```

   You may assume that the operators PLUS, TIMES, DIVIDES, MINUS are always binary.

   Reverse Polish notation is a convenient representation of expressions trees, where every operator is written behind its operands. Reverse Polish notation is defined by:

   $$\mathrm{RPOL}(\ f(t_1, \ldots, t_n)\ ) = \mathrm{RPOL}(f_1) \cdot \ldots \cdot \mathrm{RPOL}(f_n) \cdot f.$$

   For example, the expression `(1+(2*A))*` will be represented by `1 2 A * +`. Reverse polish notation useful for evaluating expressions. Expressions can be evaluated by traversing the expression from left to right using a stack. When an object is encountered, it is pushed on the stack. When an operator is encountered, the operator is performed on the two operands on the top of the stack, and the result is pushed back on the stack. Write a function

   ```
   double evaluate( const std::list< token > & expr,
                    const listconst varstore& );
   ```

   that evaluates expressions in Polish notation. You can use the tokenizer from the previous exercise to read `expr`.

2. Polish notation is similar to reverse Polish notation, but in Polish notation every operator comes before its arguments. The formal definition is

   $$\mathrm{POL}(\ f(t_1, \ldots, t_n)\ ) = f \cdot \mathrm{POL}(f_1) \cdot \ldots \cdot \mathrm{POL}(f_n).$$

   Write a function

[1] I that by now that the right word is 'FACTORIAL' but it is too late to change it now

```
   std::list< token > rpol2pol( const std::list< token > & )
```

that converts reverse Polish notation to Polish notation.

3. Write a function

```
   std::ostream& operator << ( std::ostream& ,
                               const std::list< token > & pol );
```

that prints an expression in Polish notation, but the output must be in standard infix notation. The function must insert parentheses where needed, but not more parentheses than needed. So if the input list is `+, *, 1, 2, 3`, the output will be `1 * 2 + 3`. If the input list is `* + 1 2 3`, then the output will be `(1 + 2 ) * 3`. (The best way to do this, is by using an additional function

```
void printinfix( std::ostream&,
                 std::list< token > :: const_iterator & p,
                 unsigned int leftattr,
                 unsigned int rightatttr );
```

Here `leftattr` is the attraction of the left context, and `rightattr` is the attraction of the right context. Parentheses must be inserted when the attraction (left or right) is bigger than the attraction of the main operator that we try to print.