# Expression Evaluation

Date assigned: Thursday, October 8, 2009
Date due: Tuesday, October 19, 2009
Points: 50

The basis for this problem is taken from Computer Science by J. Stanley Warford.

The purpose of this assignment is to allow the user the ability to enter any numeric expression in infix form. You will then convert the expression to postfix form using the algorithm below. Finally, print out the result of evaluating the expression.

If we look at an expression such as 1+3*4-2 we can visually see that 3*4 happens first and then 1+12 happens next and 13-2 happens last. How would this work if we wanted the computer to do the evaluation?

For this assignment, we will assume the existence of the following operators and their associated precedence:

| Operator | Name | Example | Result | Precedence |
|---|---|---|---|---|
| ^ | Exponentiation | 2^3 | 8 | 3 |
| * | Multiplication | 2*3 | 6 | 2 |
| / | Division | 2/3 | 0 | 2 |
| + | Addition | 2+3 | 5 | 1 |
| - | Subtraction | 2-3 | -1 | 1 |

Note: All operators of equal precedence are left associative.

Algorithm for computing infix expressions to postfix:

1. Scan an arbitrary well-formed integer expression consisting of integer numbers and the operators defined above from left to right.
2. If the item is a number, then move it to the postfix expression.
3. If the item is an operator, compare the item with the operator on top of the stack
    a. If the item on top of the stack has a lower priority than the operator from the infix expression, push the operator from the infix expression onto the stack.
    b. If the item on top of the stack has a priority greater than or equal to the operator from the infix expression, pop items off the stack and place them in the postfix expression until the priority of the top element of the stack is less than the infix operator priority or the stack is empty. Then push the infix item operator on top of the stack.
4. After the entire infix expression has been scanned, pop any remaining operators from the stack and place them in the postfix expression.

Example:

| Postfix Output | Stack | Infix Expression |
|---|---|---|
| empty | empty | 2+3*5+4 |
| 2 | empty | +3*5+4 |
| 2 | + | 3*5+4 |
| 2 3 | + | *5+4 |

```
2 3                   +*              5+4
2 3 5                 +*               +4
2 3 5 *               +                +4
2 3 5 * +             empty            +4
2 3 5 * +             +                 4
2 3 5 * + 4           +              empty
2 3 5 * + 4 +         empty          empty
```

Note1: Numeric values will be any legal unsigned int. Do not worry about arithmetic overflow for the overall value of the expression.

Note2: Spaces can exist in expressions but not in the numbers themselves.

Note3: You are to design a stack module reusing the code from your list module. You will also need another module for converting infix expressions to postfix form.

Note4: The program is to be interactive and work exactly as follows:

```
               Infix to Postfix Evaluator

Enter infix expression:      2+3*5+4
Postfix expression is:       2 3 5 * + 4 +
Value of the expression is:  21

Continue (y/n):
```

Note5: Your code is to be written in C using Eclipse3.5 and tested on zeus. Programs written in other environments will not be graded. Submit a file called **04yourlastname.tar.gz** using the submit script discussed in class. Once extracted, this tar file must produce a project called **InfixToPostfix** that has the following folders: Headers, Sources, Includes, Binaries, Testcases, and Tarfiles. Your makefile is to be named Makefile and is to contain clean and valgrind options. Remember, I will simply drop down the tar file and run your program from the command line, so make sure to test your program accordingly.

Note6: You must use the most recent version of list.h (version 3.0) in the CS300 Public folder on zeus.

**IMPORTANT**

What I am looking for in your solution is reusability and modularity. The solution to this problem has the potential to be a work of art or a piece of junk; obviously, I'm looking for the former. How you decide to break up your solution into modules is the key. Don't start writing code until you know the number of modules you are going to need and what they are going to do.

Please start this assignment early and see me if you have questions. I do not see this assignment as very difficult conceptually, but in terms of C coding, many of you will encounter several problems.