CIS 343 – Structure of Programming Languages
Fall 2014, Wednesday, October 8, 2014

Programming Assignment #3
Infix Expression to Postfix Expression Conversion and Evaluation
Due Date: Wednesday, October 22, 2014

Problem Specification

Write a program in C language to perform the following two functions: 1) convert a given infix arithmetic expression to a postfix arithmetic expression and evaluate it, and 2) evaluate a given postfix arithmetic expression.

You can assume that a valid expression (infix or postfix) is entered as input to the program. Also, assume that an infix expression can contain only integers, binary operators (+, -, *, /, %, ^), '(', and ')'. Similarly, a postfix expression can contain only integers and binary operators. Each token (integers, binary operators, or parenthesis) in infix or postfix expression is separated by at least one space.

You are supplied with `stack.h` and `stack.c` files containing functions that implement a stack using a linked list. You are required to use these functions in your implementation. The header file `infix2postfix.h` contains a set of function declarations/prototypes that you must implement. Complete the implementation of these functions in `infix2postfix.c` file. You are also provided with `driver.c` file that contains the `main()` function to test your implementation of functions in `infix2postfix.c` file.

Use the following algorithms in your program. Each algorithm uses a stack in support of its operations, and in each algorithm the stack is used for a different purpose. The `stacktest.c` file contains sample code that shows how to use the functions in `stack.h` file.

The algorithm for converting an infix to postfix:

```
Read infix expression from left to right.
While there are more tokens in the infix expression, do the
following:
```

- If the current token is an operand (integer), append it to postfix expression.
- If the current token is a left parenthesis, push it onto the stack.
- If the current token is an operator:
  - Pop operators (if there are any) from the stack with "stack precedence" equal or higher than the "input precedence" of the current token, and append popped

operators to the postfix expression (see below for information on stack/input operator precedence).
    - Push the current token onto the stack.
- If the current token is a right parenthesis:
    - Pop operators from the stack and append them to postfix expression until a left parenthesis is at the top of the stack.
    - Pop (and discard) the left parenthesis from the stack.

Pop the remaining operators from the stack and append them to postfix expression.

The algorithm for evaluating a postfix expression:

Read postfix expression from left to right.
While there are more tokens in the postfix expression, do the following:

- If the current token is an operand (integer), push it onto the stack.
- If the current token is an operator:
    - Pop the top element into variable y.
    - Pop the next element into variable x.
    - Perform "x operator y".
    - Push the result of the calculator onto the stack.

Pop the top value of the stack. This is the result of evaluating the postfix expression.

Input and Stack Precedence of Operators

Operator	Input Precedence	Stack Precedence	Associativity
+ -	1	1	Left-to-Right
* / %	2	2	Left-to-Right
^	4	3	Right-to-Left
(	5	-1	

Sample Execution

```
$ ./driver
(1) Convert Infix to Postfix Expression
(2) Evaluate Postfix Expression
(3) Quit
Enter selection (1, 2, 3): 1
Enter Infix Expression: 3 * ( 4 - 2 ^ 5 ) + 6
    Postfix: 3 4 2 5 ^ - * 6 +
    Value: -78
```

```
Enter selection (1, 2, 3): 1
Enter Infix Expression: 3 ^ 2 ^ ( 1 + 2 )
    Postfix: 3 2 1 2 + ^ ^
    Value: 6561

Enter selection (1, 2, 3): 2
Enter Postfix Expression: 3 2 1 2 + ^ ^
    Value: 6561

Enter selection (1, 2, 3): 2
Enter Postfix Expression: 3 4 2 5 ^ - * 6 +
    Value: -78

Enter selection (1, 2, 3): 3
Bye.
```

Deliverables

1. Upload the source file (`infix2postfix.c`) on Blackboard by midnight on due date.
2. I will use the submission date/time on Blackboard as your official submission date/time.
3. It is your responsibility to make sure the submission on Blackboard went through successfully.
4. Because of possible portability issues, make sure your program compiles and runs on EOS machines before submitting any source file(s) on Blackboard. I will compile, run, and test your program on EOS when grading.
5. Place the supplied `makefile` in the same folder as the other source files. You can compile and run the program by simply typing this command:
   ```
   $ make
   $ ./driver
   ```
6. Late penalty (10% per day) applies after Wednesday, October 22nd.