COL 100M - Lab 11

Week of April 8, 2018

1 Instructions

1. Use the OCaml top-level to develop and debug your code.
2. Please write the code in a file first and then test your code in the OCaml top-level using the directive *#use "foo.ml"* for the file *foo.ml*
3. You may assume that all inputs are valid unless otherwise stated in the problem.
4. In questions that require string outputs, be careful not to include leading or trailing whitespace.
5. You may submit and evaluate your code a *maximum* of 15 times without penalty. Subsequently, you will lose 2 marks per additional evaluation. Therefore, please ensure that you have thoroughly debugged your code before submitting.

The following submission files are *required*:

1. **in_to_post.ml**
2. **evaluate.ml**
3. **big_eval.ml**

2 Learning

2.1 Checking validity of an expression

Given an expression string, the following algorithm examines whether the order of '(' and ')' is correct. If it is correct, we call that a valid infix expression.

Algorithm:

1. Declare a stack.
2. Scan the expression string from left to right.
   - If the current character is '(', push it on the stack.
   - If the current character is ')', pop the top character from stack if it is not empty; else if the stack is empty, the parenthesis are not matched and the expression is not valid.
   - If the current character is anything else, skip it.
3. Repeat the above step till scanning is over.
4. After iterating over the entire expression, if the stack is not empty or contains some '(', then the expression is not valid otherwise the expression is valid.

2.2 Converting infix expression to postfix expression

This algorithm assumes a valid infix expression as an input and outputs the corresponding postfix expression.

Algorithm:

1. Declare a stack.
2. Scan the infix expression from left to right.
   - If the current character is a '(', push it to the stack.
   - If the current character is an operand, output it.
   - If the current character is ')', pop and output from the stack until a '(' is encountered; Pop the '(' encountered; Do not output the parenthesis.
   - If the current character is an operator
     - If the precedence of the current operator is greater than the precedence of the operator in the stack(or the stack is empty), push it on the stack.
     - Else, pop higher or equal-precedence operators from the stack to the output; stop before popping a lower-precedence operator or a '('. Push the scanned operator to the stack.
3. Repeat the above step till the entire expression is scanned.
4. Pop the remaining operators from the stack to the output postfix expression.

2.3 Evaluating a postfix expression

The following algorithm evaluates a valid postfix expression without any parenthesis.

Algorithm:

1. Declare a stack.
2. Scan the postfix expression from left to right.
   - If the current character is an operand, push it onto stack.
   - If the current character is an operator, pop top elements (operands) from stack based on number of operands required by the operator. Eg. for binary operator, pop top 2 elements in order x then y and perform z = y operator x . Push z onto stack.
3. Repeat the above step till scanning is over.
4. After scanning the complete postfix string, print not valid if stack has more than 1 operand otherwise pop the result from the stack and print it.

3 Assignment

3.1 Functionality

3.1.1 Infix to Postfix expression conversion

The main aim of this assignment is to evaluate an infix expression. For this purpose, we first convert expression in infix form to postfix as the postfix form is entirely unambiguous and is much easier to translate to a format that is suitable for direct execution.
This week you will make a file **in_to_post.ml** which converts a parenthesised infix expression to an postfix expression. Given an infix string A, print the postfix string if the infix string is valid otherwise print "not valid". You have to read the input from the standard input and print the output string without extra leading or trailing spaces. Use stack to convert an expression from infix to postfix notation. **Assume the standard precedence order of operators**, if expression is not parenthesised.

3.1.2 Evaluation of expressions

Make a file **evaluate.ml** which will use file in_to_post.ml (from 2.1.1). Given an infix string A, compute its postfix expression using previous file and if the input string is valid evaluate the postfix expression and print the answer value by converting float (truncate the answer to 4 decimal places) to string, otherwise print "not valid" if the input is not valid. Use stack for evaluating postfix expression. Operators in expression include +, -, *, /, ∧ and logarithms(assume natural logarithm). All these operators work with float as well as integer operands.

3.1.3 Evaluation of BigInt expressions

Make a file **big_eval.ml** which will use in_to_post.ml (from 2.1.1). Given an infix string A, compute its postfix expression using previous file and if the input string is valid evaluate the postfix expression and print the answer value. It is similar to 2.1.2 except the fact that the numbers here will be of BigInt type which you have implemented in BigInt module in a previous assignment and operators supported here are +, -, *, / only.

3.2 How your program will be run

Your program will be run from the *command line* after compiling it into an executable. That is, your program will be compiled using the following command: **`$ ocamlc -o in_to_post in_to_post.ml`**. The executable `in_to_post` is then run using the command: **`$ ./in_to_post < inFile`**, where `inFile` is a text file whose format is described in the next section. Note that, your program should consist of a *main* function that will read the contents of the `inFile` and then perform the operations as specified.

3.3 Input

The input to your program will be a file called inFile that will consist of multiple lines, each with the following format:

- For 2.1.1 and 2.1.2: Each line in the input file will be an expression in infix notation.
- For 2.1.3: Each line in the input file will be an expression involving BigInt numbers in the infix notation.

Each line in the input will have space separated operators, numbers and parenthesis.

3.4 Output

- For 2.1.1: Print the postfix expression in string format or "not valid" string if input string is not valid, one in each row. The operators and the numbers should be printed in a space-separated format.
- For 2.1.2: Print the evaluated value of expression in string format(truncated to 4 decimal places)or "not valid" string if input string is not valid, one in each row.
- For 2.1.3: Print the evaluated value of the BigInt expression in string format or "not valid" string if input string is not valid, one in each row.

Sample Input File for 2.1.1 and 2.1.2

(12 + 3.2) * 5.44
12 / ((3 + 0.2) ∧ 5)
(39 + (3 / log (10)))

Sample Output for 2.1.1

12 3.2 + 5.44 *
12 3 0.2 + 5 ∧ /
not valid

Sample Output for 2.1.2

82.688
0.0357
not valid

Sample Input File for 2.1.3

(1111111111111111111 + 22222222222222222222) * 100000000000000000000 + 1
55555555555555555555 + 22222222222222222222 * 100000000000000000000 + 3
77777777777777000000000000000000 / 1000000000000000 - 1
(1111111111111111111 + 22222222222222222222 * 100000000000000000000 + 1

Sample Output for 2.1.3

333333333333333333330000000000000000000001
22222222222222222222555555555555555555558
777777777777776
not valid