Middle East Technical University | Department of Computer Engineering

# CENG 242

Programming Language Concepts

Spring '2012-2013

Homework 1

Due date: 10 March 2013, Sunday, 23:55

## 1 Objective

In this homework, you will be given a string containing infix expressions. You have two objectives in this homework. The first one is converting the infix expressions to postfix expressions. You will be using Dijsktra's Shunting Yard Algorithm to convert infix expressions to postfix expressions. This is an algorithm that makes use of stacks to convert expressions. The details of the Shunting Yard algorithm are provided below. The second objective is evaluating the postfix expressions and providing the result of the expressions. The details of the algorithm that you will be using at this part are, also provided below.

## 2 Specifications

- The expressions will be constructed from only **one digit integers** and '+','-','*' and '/' operations. (For example "2*2+4/2" or "2-9/3+6" or "1*2*3-5" etc.).
- "`data Stack = Empty | Element Char Stack deriving Show`" This is the stack data structure that you will be using while implementing Shunting Yard Algorithm and evaluation algorithm. Of course, you are free to define further data structures if you need.
- You will write two functions the first one infixToPostfix in the form: "`infixToPostfix` :: [`Char`] $->$ [`Char`]" This means that it will take one input which is the expression list. And the result will be character list which will hold the postfix expression.
- And evaluatePostfix will be in the form: "`evaluatePostfix` :: [`Char`] $->$ `Float`" which means that it will take a character list (postfix expression) as input and it will return a single float value as output (result of the expression).
- Your functions **must have above mentioned types**. To be sure you can use :t function at hugs to see the type of the functions you write.
- You can assume that there will **only be positive one digit integers** in the input but at the **intermediate steps of the evaluation part you may face negative or fractional numbers** you should make your implementations accordingly.
- You can assume that there **won't be any spaces in the input string.**

- Precedence of '*' and '/ 'are same and higher than precisions of '+' and '-'. Precedence of '+' and '-' are also same. Also note that all of the '+', '-' , '*' and '/' characters are left associative.
  The pseudo codes of the both algorithms are provided below:
  **Shunting Yard Algorithm**

```
Start: dequeue token from input
if operand (number) then
    add to output queue;
end
if the token is an operator, o1 then
    while there is an operator token, o2, at the top of the stack do
        if o1 is left-associative and its precedence is less than or equal to that of o2 or
        o1 has precedence less than that of o2, then
            pop o2 off the stack, onto the output queue;
        end
    end
    push o1 onto the stack;
else
    return to start as long as tokens remain in input;
    pop remaining operators from stack and add to output queue;
end
```

**Algorithm 1:** Shunting Yard algorithm

**Postfix Evaluation Algorithm**

```
while there are input tokens left do
    Read the next token from input.
    if the token is a value then
        Push it onto the stack.;
    else
        (Token is a operator)
        (It is known a priori that the operator takes n arguments) if there are fewer
        than n values on the stack then
            Error The user has not input sufficient values in the expression.
        else
            Pop the top n values from the stack;
            Evaluate the operator, with the values as arguments;
            Push the returned results, if any, back onto the stack;
        end
    end
end
if there is only one value in the stack then
    That value is the result of the calculation;
else
    Error The user input has too many values;
end
```

**Algorithm 2:** Postfix Evaluation Algorithm

The algorithms are taken from Wikipedia, you are advised to go and make further reading from these pages:
1. Shunting Yard Algorithm
2. Reverse Polish Notation
at Wikipedia.

# 3 Examples

The working of Shunting Yard Algorithm on an input: The working of Postfix Evaluation


Figure 1: Shunting Yard Algorithm Example

Example with the output of the first function:

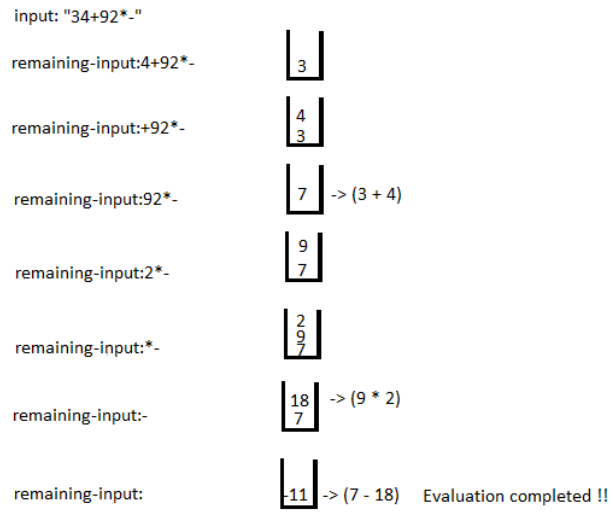


Figure 2: Postfix Evaluation Example

As a result, for this input your functions should work as the following:

>> `infixToPostfix` "$3 + 4 - 9 * 2$"
>> "$34 + 92 * -$" (This is the output that should be returned from your function)
>> `evaluatePostfix` "$34 + 92 * -$"
>> $-11$ (This is the output that should be returned from your function)

# 4 Submission

Submission will be done via COW. You should upload a single Haskell file called "$hw1 - e1234567.hs$". The **name of the file should not have any spaces**. Here e1234567 indicates your student ID you should fill it accordingly.
Besides, **you should test your codes in inek machines with hugs** before submitting. Since black box method will be used in evaluation, be careful about the name of functions, data structures etc.
Late submission: At most 3 late days are allowed. After 3 days, you get 0.

# 5 Grading

This homework will be graded out of 100. The infixToPostfix functions grade is 60/100 and evaluatePostfix functions grade is 40/100.

# 6 Cheating Policy

**We have zero tolerance policy for cheating.** People involved in cheating will be punished according to the university regulations.
Both parties involved in cheating get 0 from all of the 6 homework's and will be reported to the university's disciplinary actions committee.