Project 4: RPN Calculator

dr Szymon Murawski

The aim of this project is to get familiar with stacks and queues data types by writing a simple Reverse Polish Notation (RPN) calculator. This project is also perfect for trying to write it using TDD (Test-driven development) technique. If you are willing to give it a try i highly recommend you to read about NUnit - it is a testing framework for C#. Introduction to NUnit will also be a topic of one of the laboratories, so do not worry, if you have troubles reading online tutorials.

Task 1: *RPN calculator*

RPN calculator operates on postfix operator notation, meaning that operators are written after operands. This notation makes it very easy to calculate equation, by simple stack usage. In this task we want to write a calculator, that will solve a postfix equation. For details on implementation refer to lecture notes (Lecture 5 - Linear Data Structures).

1. Create your own stack class. As underlying data structure use simple array of size 10. Your stack class should feature pop, push and isEmpty operations. Refer to the lecture notes on details of implementation. Make your stack hold floating point (double) values, but if you can make your stack generic, it will come in handy later!

2. Implement a calculator (either as a function or as a class) that will solve simple postfix equation. As an input it will be given single string - postfix equation and that function should return floating point number (double). For starters make it deal with simple equations of only two operands and one operator. Some test cases:

   - $2\,3\,+ = 5$
   - $5\,4\,* = 20$
   - $4\,2\,/ = 2$
   - $11\,3\,- = 8$
   - $6\,2\,\wedge = 36$

3. Expand the function from previous point, so that it can take many operands and operators. Some test cases:

   - $3\,1\,7\,+\,* = 24$
   - $5\,3\,8\,12\,+\,*- = 55$
   - $4\,2\,+\,9\,* = 54$
   - $3\,1\,-\,7\,* = 14$

4. Expand the function even more, so that it can throw an exception when as an input its given improper postfix equation, like so:

   - Empty input - throw "No input given" exception
   - "3 3" - throw "Wrong equation: not enough operators" exception

- "3 3 + +" - throw "Wrong equation: too many operators" exception
- "1 2 3 4 5 6 7 8 9 10 11 + * + * + * + * + * + *" - throw "Stack overflow" exception

Task 2: *Infix to postfix conversion*

Now that we have working RPN calculator we can try working on algorithm, that will convert infix equations to postfix ones.

1. We will need a stack, that can contain chars. Either make your stack from previous point generic, or make another stack class that will hold char values.
2. Write a function for converting infix to postfix equations. As an input the function should take single string - infix equation, as output it should also give single string - postfix equation. First make the function work for simple cases of two operands and one operator:
   - $2 + 3 \rightarrow 2\,3\,+$
   - $6 - 2 \rightarrow 6\,2\,-$
   - $12 * 5 \rightarrow 12\,5\,*$
   - $3 \wedge 2 \rightarrow 3\,2\,\wedge$
3. Expand the function, so now it can deal with multiple operators of the same precedence
   - $2 + 3 - 2 \rightarrow 2\,3\,+\,2\,-$
   - $5 * 4/9 \rightarrow 5\,*\,4\,9\,/$
4. Now make it deal with operators with different precedence
   - $2 + 3 * 5 \rightarrow 2\,3\,5\,*\,+$
   - $5 * 3 + 1 \rightarrow 5\,3\,*\,1\,+$
   - $2 * 3 + 8/4 \rightarrow 2\,3\,*\,8\,4\,/\,+$
5. Add parenthesis
   - $7 * (2 + 3) \rightarrow 7\,2\,3\,+\,*$
   - $2 * (3 + 8)/4 \rightarrow 2\,3\,8\,+\,*\,4\,/$
   - $9/((4 + 2) * 3) \rightarrow 9, 4\,2\,+\,3\,*\,/$
6. Enable $\wedge$ operator:
   - $2 \wedge 3 \wedge 4 \rightarrow 2\,3\,4\,\wedge\,\wedge$
   - $4 * 9 \wedge 6 \rightarrow 4\,9\,6\,\wedge\,*$