**Calculator  
Learn about Reverse Polish Notation by building a simple calculator**

TEACHER’S GUIDE

Created by Richard Pawson

[Creative Commons License](https://creativecommons.org/licenses/by-sa/4.0/)  
This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

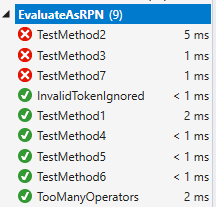
# Introduction

# Exercise 1: Exploring the out-of-the-box solution

Paste in a partial screenshot showing the whole calculator after entering the expression above and hitting evaluate.

# Exercise 2: Get the RPN Calculator working

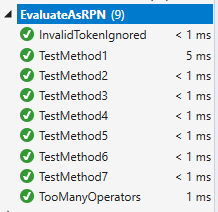
Make the changes and check that you understand the general idea of the code. Then run the tests. Some of the tests under EvaluateAsRPN will now pass, but some will still fail. Paste in a partial screenshot showing which of the EvaluateAsRPN tests pass, and which fail.



Double click on each test to examine the test scenario in code. Find the expressions that it is failing on and write them into the document also.

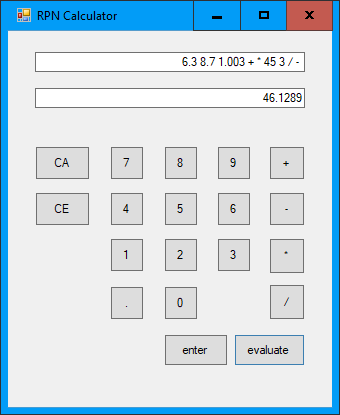
A: It is failing on the minus and divide operations.

Verify that all the EvaluateAsRPN tests pass (the other sets of tests will still all fail for now) and paste in a partial screenshot showing this.



Now you can run the Calculator itself. Try out a fairly complex (but sensible) expression and paste in a screenshot showing the calculator displaying the full expression and the result.

For example:



Finally, while we’re still thinking about RPN, go back to the explanation of the algorithm a couple of pages back and learn it. Then use that algorithm *manually* (without the aid of the calculator) to evaluate the following RPN expression:

7 6 3 4 + \* -

*Show your working* by sketching a stack and showing the values on the stack at each step. (This is useful practice for a possible exam question).

Token processed:

7 6 3 4 + \* -

Stack after each step:

7 6 3 4 7 42 -35 (answer)

7 6 3 6 7

7 6 7

7

### Complete code for Core.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Calculator

{

public class Core

{

private List<object> Tokens = new List<object>();

public void Clear()

{

Tokens = new List<object>();

}

internal void AddSymbolAsToken(char symbol)

{

Tokens.Add(symbol);

}

public double AddNumberAsToken(string numberAsText)

{

double number = Convert.ToDouble(numberAsText);

Tokens.Add(number);

return number;

}

public string TokensAsString()

{

var sb = new StringBuilder();

foreach (var token in Tokens)

{

sb.Append(token.ToString()).Append(" ");

}

return sb.ToString();

}

public double EvaluateTokensAsRPN()

{

return EvaluateAsRPN(Tokens);

}

public static double EvaluateAsRPN(List<object> Tokens)

{

double result = 0;

var stack = new Stack<double>();

foreach (object token in Tokens)

{

if (token is double)

{

stack.Push((double)token);

}

else

{

switch ((char)token)

{

case '+':

stack.Push(stack.Pop() + stack.Pop());

break;

case '-':

var b = stack.Pop();

var a = stack.Pop();

stack.Push(a - b);

break;

case '\*':

stack.Push(stack.Pop() \* stack.Pop());

break;

case '/':

var d = stack.Pop();

var c = stack.Pop();

stack.Push(c / d);

break;

}

}

}

result = stack.Pop();

return result;

}

public double EvaluateTokensAsInfix()

{

var tokensAsRPN = ConvertInfixToPostfix(Tokens);

return EvaluateAsRPN(tokensAsRPN);

}

public static List<object> ConvertInfixToPostfix(List<object> inputTokens)

{

throw new NotImplementedException();

}

public static int Priority(char c)

{

throw new NotImplementedException();

}

}

}