**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

# Model answers

# 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)

{

var s = new Stack<char>();

var outputList = new List<object>();

foreach (var t in inputTokens)

{

if (t is double) //token is a value

{

outputList.Add(t); //send it straight to the output

}

else

{

char token = (char)t; ///... so cast it to a char

if (token == '(')

{

s.Push(token);

}

else if (token == ')')

{

while (s.Count != 0 && !s.Peek().Equals('('))

{

outputList.Add(s.Pop());

}

s.Pop();

}

else

{

while (s.Count != 0 && Priority(s.Peek()) >= Priority(token))

{

outputList.Add(s.Pop());

}

s.Push(token);

}

}

}

while (s.Count != 0) //Unload any remaining operators onto the stack

{

outputList.Add(s.Pop());

}

return outputList;

}

public static int Priority(char c)

{

if (c == '\*' || c == '/')

{

return 2;

}

else if (c == '+' || c == '-')

{

return 1;

}

else return 0;

}

}

}