# Lab: Implement a console application to calculate the output from set of instructions.

## Scenario

Instructions comprise a keyword and a number that are separated by a space per line. Instructions are loaded from file and results are output to the screen. Any number of Instructions can be specified.

Instructions can be any binary operators of your choice (e.g., add, divide, etc.), ignoring mathematical precedence. The last instruction should be "apply" and a number (e.g., "apply 3"). The calculator is then initialized with that number and the previous instructions are applied to that number.

## Objectives

Demonstrate and show case your C# development skills;

* SOLID principles, Single responsibility principle, Interface design, Open closed principle.
* Dependency Injection, C#, NUnit Test Framework, Fluent Validations
* TDD, ability to write unit tests.

#### 

**Setup**

The solution file exists under ..\Calculator\Calculator\Calculator.sln. The project was built and created using VS 2019.

The solution contains 3 projects;

1. A console application project Calculator.csproj
2. The engine class library Calculator.Engine.csproj
3. The unit test project Calculator.Engine.Tests.csproj

The solution requires the following NuGet packages

1. Microsoft.Extensions.DependencyInjection
2. FluentValidation.AspNetCore

The console application contains the text files listed below they are marked as copy to output.

When the console application is Run / Launched, the user can provide the path to one of the files listed.

|  |  |
| --- | --- |
| 1 | ..\Calculator\Calculator\Files\Calc01.txt |
| 2 | ..\Calculator\Calculator\Files\Calc02.txt |
| 3 | ..\Calculator\Calculator\Files\Calc03.txt |
| 4 | ..\Calculator\Calculator\Files\Calc04NoApply.txt |
| 5 | ..\Calculator\Calculator\Files\Calc05EmptyFile.txt |
| 6 | ..\Calculator\Calculator\Files\Calc06MissingOperation.txt |

**Summary**

The Calculator.csproj has a Program.cs file and uses dependency injection to talk to the Calculator engine libraries.

I have used Fluent validation to perform the necessary validation. To build the map between the instructions and the mathematical operations, the application uses regular expressions.

The regular expressions can be found at ..\Calculator\Calculator.Engine\Extensions.cs

The Calculator.Engine project applies the generic IProvide interface, it is implemented as 3 provider classes CalculatorEngineProvider.cs, EvalProvider.cs and InstructionToExprProvider.cs

The unit tests project will test each of the providers above

C# does not contain an eval function like javascript, however there is a Compute method within the DataTable library which performs an eval.