How Will my project will be marked?

This project counts for 15% of your final grade and will be graded using the following grid:

	Task	Mark
1	De-serialize CSV input to C# List named InFix (CSVFile class)	2
2	Convert Infix expressions stored in the InFix list to prefix expressions and save them in a generic list named PreFix **https://tinyurl.com/y2wxf9nw**	2
3	Convert Infix expressions stored in the InFix list to postfix expressions and save them in a generic list named PostFix **https://tinyurl.com/y7gs8yvo**	2
4	Evaluate each expression in the PreFix list using Expression Tree **https://tinyurl.com/y2kt2p6t**	2
5	Evaluate each expression in the PostFix list using Expression Tree. **https://tinyurl.com/y54f4tf9**	2
6	Use the IComparer interface to compare the results from Prefix and Postfix evaluation. Note: Prefix and Postfix evaluation must match.	1
7	Generate an XML file with the structure as below using **extension methods by extending StreamWriter for each method as defined in the Functional Requirements Section.** <root> <elements> <sno>1</sno> <infix>2+3</infix> <prefix>+23</prefix> <postfix>23+</postfix> <evaluation>5</ evaluation > <comparison>true</comparison> </ elements > < elements > ……. …….. </ elements > …… ……. </root> **Note: sno refers to the serial or expression number as provided in CSV file under 'sno'**	2
8	Creative console interface to display the outputs after each conversion or evaluation step in addition to the summary report	2
9	Proper submission and suitable comments	1
10	Submit Video	4
	Total	**20**

Project Description

We are used to seeing any mathematical expressions in infix notation form; that is, the operator comes in between the operands. There is a clear distinction of how to separate which operator proceeds. Moreover, parentheses also make it easier for humans to carry out the necessary evaluation. On the contrary to humans, machines interpret expressions differently. Prefix and postfix notations may be used varyingly for a machine to understand any expression. This is because of the simplicity in parsing them.

Infix Notation: It is most commonly used to represent a mathematical expression. The operators in such notation are placed in between operands. For example, **(a-b) *c**.

Prefix (Polish) Notation: In this notation, the operator proceeds the operands. One of the advantages of using prefix notation is, it gets rid of parenthesis and is easier to parse than infix notation. Prefix notation for (a-b) *c is represented as ***-abc**

Postfix (Reverse Polish) Notation: As the name suggests, it's like a reverse of polish or infix notation in the sense that operators are proceeded by operands. Postfix notation for (a-b) *c is represented as **ab-c***

Attached with this document is a CSV file containing infix expressions. This project's primary goal is to convert Infix notation expression into postfix and prefix notation and evaluate them using Expression Trees.

Functional Requirements:

1. Create a class called CSVFile. Within this class, create a method called CSVDeserialize to dump the content of the CSV file to a list.
2. Each conversion process should be encapsulated within a public class. Choose an appropriate name for each class.
3. Create a class called ExpressionEvaluation. This class will include the necessary methods to implement the evaluation processes.
4. Build a class Called XMLExtension that would have the following extension methods:
    a. WriteStartDocument: This method should include xml version and encoding
    b. WriteStartRootElement: This method should add the **root** tag to the file
    c. WriteEndRootElement: This method should end the **root** tag.
    d. WriteStartElement: This method should add the **element** tag to the file
    e. WriteEndElement: This method should end the **element** tag
    f. WriteAttribute: This method should add each attribute with its values.

5. Create a class called “CompareExpressions” that would inherit the IComparaer interface. Within this class, override the Compare method to compare the results out of the conversion processes.
6. All provided web links are provided to help you in implementing the necessary conversion processes. It is your responsibility to verify and update it as needed to get the expected matched outputs.
7. The provided links do not use expression trees to evaluate the mathematical expressions. You need to update the pseudocode out there to work along with the expression tree. You need to build a binary expression that would handle two numbers. Such an expression tree performs the required calculations based on the four operators (e.g., + or -) and returns the result as a string; we did a simple example similar to that approach within the class (in lambda expression week). It would be much better to build this tree within a method. This method will be used to evaluate any prefix and postfix expression. You can use any online source that may help to do this part.
8. The application must write to the console the results after implementing each conversion or evaluation process. Also, it must provide a summary report, including all conversion outputs, similar to the provided sample one.
9. After demonstrating the summary report in the console, the application must prompt the user to upload the required XML file and bring it up on any web browser.
10. The provided text file will not be used in your developed application. It’s just a second reference to verify the CSV file equations because some Excel versions may mix up those equations and convert them into wrong formulas.

Other Requirements and Notes:

The following is a list of requirements and constraints for your application:

1. Visual Studio and .NET have tools that can be used with this project.
2. Include detailed comments in your code describing the critical aspects of your program.
3. Each class must be defined in a single separate cs file.
4. You can add what you think fits the application.
5. You are **not** allowed to edit the source data file.
6. Create a “Data” folder in your project to store the source data file and resulting XML file.

How should I submit my project?

Electronic Submission:

Please submit the entire solution as a single zipped file to the submission folder "Project on FOL"
. before the due date: Sunday, April 6, 11:59 PM
Late submission policy: Only a one-day late submission is allowed, with a 20% penalty on the
. total project mark
. Submission format: Name the zipped file as "Project2_Group_X", where X is your group number
Submission responsibility: Only one group member needs to submit the solution. All team members will receive the same mark unless a complaint is raised about a non-cooperative member. If verified, the non-cooperative member's grade may be reduced to half of the overall
. team grade
Plagiarism policy: Any copied work will not be marked, and Fanshawe regulations will apply in
. such cases
. Incorrect submissions will not be marked
.Ensure that you follow these guidelines carefully to avoid penalties

The Sample output

The Summary report

Summary Report

Sno	Infix	PosFix	Prefix	Prefix Res	PostFix Res	Match
1	5+4	54+	+54	9	9	True
2	4-3	43-	-43	1	1	True
3	2*4	24*	*24	8	8	True
4	8/2	82/	/82	4	4	True
5	3/2	32/	/32	1.5	1.5	True
6	9+9-8	99+8-	-+998	10	10	True
7	9-9*3	993*-	-9*93	-18	-18	True
8	4*5/2	45*2/	/*452	10	10	True
9	7*7*2-9	77*2*9-	-**7729	89	89	True
10	5*2+8/2	52*82/+	+*52/82	14	14	True
11	9*6*2/2+9	96*2*2/9+	+/**96229	63	63	True
12	5*1-2	51*2-	-*512	3	3	True
13	6/3*5/2*8*3	63/5*2/8*3*	**/*/635283	120	120	True
14	9-3+8-2/2*4	93-8+22/4*-	-+-938*/224	10	10	True
15	9-2+7/2*8-6*7	92-72/8*+67*-	-+-92*/728*67	-7	-7	True
16	(7-3)*9+6	73-9*6+	+*-7396	42	42	True
17	7*2+(3*3)	72*33*+	+*72*33	23	23	True
18	(7+9+6)*5-(2*2)	79+6+5*22*-	-*++7965*22	106	106	True
19	(3*7+4)-4+1*2+(6/3)	37*4+4-12*+63/+	++-+*3744*12/63	25	25	True
20	8*9-5/1+9*(1-3)	89*51/-913-*+	+-*89/51*9-13	49	49	True
21	(9*8)+(5*3)	98*53*+	+*98*53	87	87	True
22	8*3-6+8-2+(2-1)	83*6-8+2-21-+	+-+-*83682-21	25	25	True
23	9-6+8*2+8-3	96-82*+8+3-	-++-96*8283	24	24	True
24	8*6-5*8+3*2/1	86*58*-32*1/+	+-*86*58/*321	14	14	True
25	9*9/3+5/5*5	99*3/55/5*+	+/*993*/555	32	32	True

The XML file

```
<?xml version="1.0" encoding="UTF-8"?>
- <root>
  - <element>
      <sno>1</sno>
      <infix>5+4</infix>
      <prefix>+54</prefix>
      <postfix>54+</postfix>
      <evaluation>9</evaluation>
      <comparison>True</comparison>
    </element>
  - <element>
      <sno>2</sno>
      <infix>4-3</infix>
      <prefix>-43</prefix>
      <postfix>43-</postfix>
      <evaluation>1</evaluation>
      <comparison>True</comparison>
    </element>
  - <element>
      <sno>3</sno>
      <infix>2*4</infix>
      <prefix>*24</prefix>
      <postfix>24*</postfix>
      <evaluation>8</evaluation>
      <comparison>True</comparison>
    </element>
  - <element>
      <sno>4</sno>
      <infix>8/2</infix>
      <prefix>/82</prefix>
      <postfix>82/</postfix>
      <evaluation>4</evaluation>
      <comparison>True</comparison>
    </element>
  - <element>
      <sno>5</sno>
      <infix>3/2</infix>
      <prefix>/32</prefix>
      <postfix>32/</postfix>
      <evaluation>1.5</evaluation>
      <comparison>True</comparison>
    </element>
  - <element>
      <sno>6</sno>
      <infix>9+9-8</infix>
      <prefix>-+998</prefix>
      <postfix>99+8-</postfix>
      <evaluation>10</evaluation>
      <comparison>True</comparison>
    </element>
```