To be completed in groups of three (3) students. You can enrol in a group via Communication > Groups > Project 2 in the course menu in FOL.

How Will my project will be marked?

This project counts for 15% of your mark 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**	3
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**	3
4	Evaluate each expression in the PreFix list using Expression Tree **https://tinyurl.com/y2kt2p6t**	3
5	Evaluate each expression in the PostFix list using Expression Tree. **https://tinyurl.com/y54f4tf9**	3
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
	Total	**20**

Project Description

We are used to seeing arithmetic expressions in infix notation form in which the operator appears between the operands like 2 + 3 * 5. Applying operator precedence rules, we can easily determine the sequence in which to perform the operations. Parentheses can also be incorporated to impose a different sequence.

Machines interpret expressions differently. Either prefix or postfix notation may be used to enable a machine to evaluate 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 precedes the operands. One of the advantages of using prefix notation is that it gets rid of parentheses 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 preceded 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 goals are to convert a series of Infix notation expressions into postfix and prefix notation expressions, and then to 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 generic list.
2. Each conversion process (to Prefix and to Postfix) 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 implement the IComparer interface. Within this class, override the Compare method to compare the results out of the conversion processes.
6. The provided web links should help you implement the necessary conversion processes. It is your responsibility to verify and update the algorithms 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 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 – week 8). 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 the results to the console after performing each conversion or evaluation process. Also, it must provide a summary report, including all conversion outputs, similar to the sample report provided below.
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 1 on FOL before the due date **Monday, April 3, 11:59 PM**. Only one-day late submission is allowed and will be penalized with 20% of the total project mark. Submissions that don't compile will not be marked! Please name the zipped file: **Project2_Group_X** (where X is the group number). Only one member within the group has to submit the solution. All team members will get the same mark unless there's a complaint of a non-cooperative member! After verifying that complaint, the non-cooperative member's grade might be deducted to half of the overall team grade. Any copied work will not be marked, and Fanshawe regulations in this situation will be applied.

The Sample output

The Summary report

Sno	InFix	PostFix	PreFix	PreFix Res	PostFix Res	Match
1	7-4	74-	-74	3	3	True
2	2+6	26+	+26	8	8	True
3	3*2	32*	*32	6	6	True
4	9/3	93/	/93	3	3	True
5	7/2	72/	/72	3.5	3.5	True
6	8+8-9	88+9-	-+889	7	7	True
7	7-4*5	745*-	-7*45	-13	-13	True
8	9*2/6	92*6/	/*926	3	3	True
9	8*8*2-7	88*2*7-	-**8827	121	121	True
10	7*2+9/3	72*93/+	+*72/93	17	17	True
11	2*6*7/2+9	26*7*2/9+	+/**26729	51	51	True
12	6*1-3	61*3-	-*613	3	3	True
13	7/3*6/2*5*3	73/6*2/5*3*	**/*/736253	105	105	True
14	9-2+8-3/2*6	92-8+32/6*-	-+-928*/326	6	6	True
15	9-3+7/2*8-6*7	93-72/8*+67*-	-+-93*/728*67	-8	-8	True
16	(6-2)*9+5	62-9*5+	+*-6295	41	41	True
17	5*6+(4*4)	56*44*+	+*56*44	46	46	True
18	(6+7+8)*5-(2*2)	67+8+5*22*-	-*++6785*22	101	101	True
19	(3*6+4)-4+1*3+(8/2)	36*4+4-13*+82/+	++-+*3644*13/82	25	25	True
20	8*9-6/2+9*(2-4)	89*62/-924-*+	+-*89/62*9-24	51	51	True
21	(9*7)+(6*4)	97*64*+	+*97*64	87	87	True
22	8*2-6+8-2+(6-3)	82*6-8+2-63-+	+-+-*82682-63	19	19	True
23	9-6+8*3+8-7	96-83*+8+7-	-++-96*8387	28	28	True
24	8*7-6*7+3*2/1	87*67*-32*1/+	+-*87*67/*321	20	20	True
25	8*8/4+6/6*6	88*4/66/6*+	+/*884*/666	22	22	True

The XML file (partial output shown)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <element>
        <sno>1</sno>
        <infix>7-4</infix>
        <prefix>-74</prefix>
        <postfix>74-</postfix>
        <evaluation>3</evaluation>
        <comparison>True</comparison>
    </element>
    <element>
        <sno>2</sno>
        <infix>2+6</infix>
        <prefix>+26</prefix>
        <postfix>26+</postfix>
        <evaluation>8</evaluation>
        <comparison>True</comparison>
    </element>
    <element>
        <sno>3</sno>
        <infix>3*2</infix>
        <prefix>*32</prefix>
        <postfix>32*</postfix>
        <evaluation>6</evaluation>
        <comparison>True</comparison>
    </element>
    <element>
        <sno>4</sno>
        <infix>9/3</infix>
        <prefix>/93</prefix>
        <postfix>93/</postfix>
        <evaluation>3</evaluation>
        <comparison>True</comparison>
    </element>
    <element>
        <sno>5</sno>
        <infix>7/2</infix>
        <prefix>/72</prefix>
        <postfix>72/</postfix>
        <evaluation>3.5</evaluation>
        <comparison>True</comparison>
    </element>
    <element>
        <sno>6</sno>
        <infix>8+8-9</infix>
        <prefix>-+889</prefix>
        <postfix>88+9-</postfix>
        <evaluation>7</evaluation>
        <comparison>True</comparison>
    </element>
```