Mid-Term Exam (TEST01)- SEC-003

Student Name:_____

Date: Friday, 10th July, 2020

Duration: 2 hours (8.30 am to 10.30 am)

Marks/Weightage*: *50/25%

Submission Instructions: Be sure to read the following general instructions carefully:

This test must be completed individually by all the students.

At the start, you must name your Visual Studio 2019 solution name according to the following rule:

FirstName-LastName_COMP212_ SectionNumber _Test01

For Example: John-Smith_COMP212_Sec003_Test01

>> And your project name should be as follows: FirstName-LastName_ExerciseNumber

For Example: John-Smith_Exercise01

>> If your test has more than one exercise, then each subsequent exercise should be **added as new project** to the **same solution created above** and named as firstname-last-name_Exercise2, firstname-last-name_Exercise3 etc.

>> After you complete coding and testing, exit Visual Studio and go to solution folder, zip it up and you will get the following zip file. For example: **John_Smith_COMP212- Sec003_Test01.zip** and upload it to **TEST-01 folder** in e-centennial.

>> Apply the naming conventions for variables, methods, classes, and namespaces:

- variable names start with a lowercase character for the first word and uppercase for every other word

- classes start with an uppercase character of every word

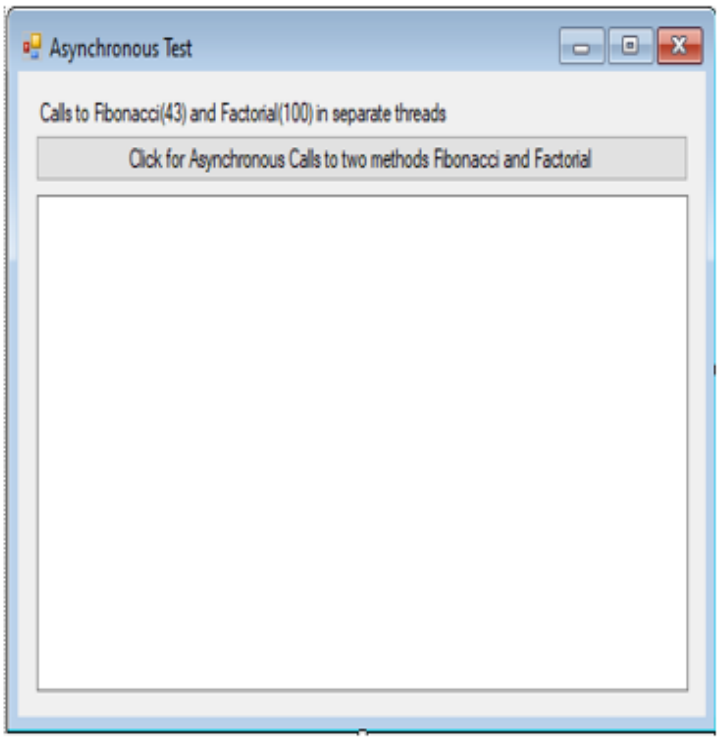
- namespace use only lowercase characters

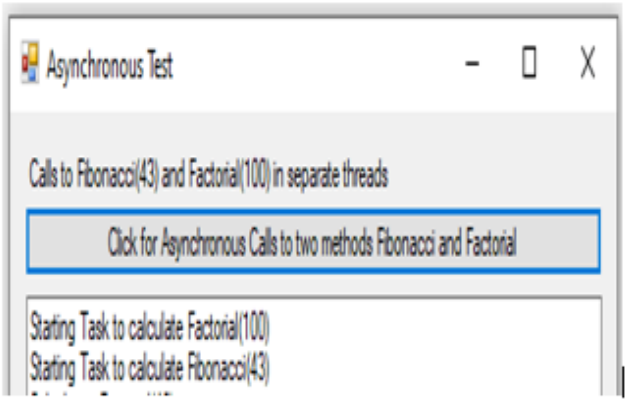
- methods start with an uppercase character for the first word and uppercase for every other word

Note: You need to maintain academic honesty and integrity. You are not allowed to email, copy, share, show your code to anyone. You must upload your test on or before the deadline, on e-centennial. No late submissions and not e-mail attachment allowed. You must implement exception handling.

Exercise 01: Asynchronous Programming. Build the following app using Win Forms. *[20 marks]*

Factorial (for 100) and Fibonacci (for 43rd term) recursive implementation should be asynchronous (use of async and await; and Task.Run() method). See the screen shot below. Refer the code example covered in the class. For large factorial values, use BigInteger type.





Exercise 02: *[10 marks]*

Create a console app and implement an extension method – int **CharCount()** for built-in class **StringBuilder** to count the number of characters contained in a StringBuilder object including blank spaces.

[You need to add a separate ***StringBuilderExtensions.cs*** file containing class – **StringBuilderExtensions,** and adding **CharCount** method in there.]

In the class file, containing Main() method, you will be testing this method.

For example, if a StringBuilder object strobj = ”This is a demo test”, the number of characters contained in strobj is 19.

Exercise 03: *[10 marks]*

Create a console C# app in which you are required to create the following generic method and test it for integers and doubles.

- public static T **SubArray**(T a[ ], int *startIndex*, int *endindex*) which returns a sub array containing the elements between - startIndex value and endindex value (but not including both).

You need to add validations for startIndex such as it should not be negative and should not be greater than size of the array. Also for endindex such as it can not be negative, less than startIndex or higher than the size of the array.

Exercise 04: Create a console app (.Net Framework) and do the following: *[10 marks]*

You are required to create the following methods using lambdas(either expression or statement lambdas).

- void **Largest**(string1, string2, string3) which displays the largest of three string values based on the length.

To test/call this method, you need to use built-in **Action<>** delegate predicate.

Second method is:

- bool **FindPrime**(int number) which displays true if the number is prime otherwise false.

To test this method, you need to use built-in **Func**<> delegate predicate

Evaluation/Rubric:

Evaluation: **Functionality**	
Correct implementation of classes (instance variable declarations, validations, constructors, properties class methods etc.)	70%
Correct implementation of test classes (declaring and creating objects, calling their methods, interacting with user, displaying results in use friendly way)	20%
Comments, correct naming of variables, methods, classes, etc. and exception handling	5%
User Friendly input/output	5%
Total	100%