Programming I – Assessed Component 1

Note: This work will focus on the material that we have covered thus far in the Programming I course. Please ensure you follow the instructions closely. Any submission that does not satisfy the core requirements such that it can be tested will be automatically scored as a marginal fail (39%) .

Further instructions and advice to help you complete this assignment will be given in lectures between now and the submission date, so make sure you attend all the lectures!

This assignment must be submitted for testing and assessment in Week 6. Further instructions on submission will be given closer to the deadline.

You will create a class/file called *FizzBuzz*. The FizzBuzz class will carry a number of methods that will be tested using our own program. As such, it is important that you ensure your method names and their signatures (return types and parameters) are exactly as described below in this specification. <u>If our testing program cannot utilise your methods (given that we expect them to be written as specified) because you have used the wrong parameter types, the wrong return types, or you have renamed and/or failed to provide the correct methods, you will fail the assignment.</u>

Part 1: FizzBuzz

For the first part, you will complete the following methods:

- **public bool IsFizz(int input)**
  - This method will check whether a given number is divisible by 9.
  - If divisible, will return true, otherwise false.
  - **Hint:** Look at the modulus operator.
- **public bool IsBuzz(int input)**
  - Similarly, this method will check whether a given number is divisible by 13 and return true/false accordingly.
- **public bool IsFizzBuzz(int input)**
  - This method will check whether the number is divisible by <u>both</u> 9 and 13 and return accordingly.

Part 2: Primes

Next add the following method:

- **public bool IsPrime(int input)**
  - This method return true if the input is a <u>prime number</u>, i.e. it is only divisible by itself and one.

Part 3: Testing

For the purposes of testing, it is expected that you will maintain a count of the number of Fizz's, Buzz's, FizzBuzz's and prime numbers encountered during any testing 'phase'. As such, a number of methods must be introduced:

- **public void BeginTesting()**
  - Clears any variables storing the total number of Fizz's, Buzz's, FizzBuzz's and prime numbers found.

- **public int TotalFizz()**
  - Returns the total number of **Fizz(int)** calls that returned true.

- **public int TotalBuzz()**
  - Returns the total number of **Buzz(int)** calls that returned true.

- **public int TotalFizzBuzz()**
  - Returns the total number of **FizzBuzz(int)** calls that returned true.

- **public int TotalPrime()**
  - Returns the total number of **Prime(int)** calls that returned true.

Note that this may require you to change the body of methods from Part 1 and Part 2 in order to count whether a call successfully returned true.

Testing in Clinics

The assessed component will be tested using code that is written by the module team. To do this, we will be testing your code using batches of randomly generated integers, the quantity of which will also be randomly generated. As such, please ensure that your code is capable of handling these numbers effectively.

General Marking Guidelines

Please refer to the assignment criteria document currently on Course Resources for a refresher on the marking scheme for this module. Note that as you move up between grade bands, all preceding requirements are expected to be satisfied (i.e. you cannot achieve a *good* submission if the *satisfactory*, *marginal fail* or *fail* are not exhibited, regardless of particular merits of your work). Determining where your work fits within these gradebands is at the discretion of the marking tutors.

To achieve an excellent (70%+) submission:

- Satisfied 90% or more of test cases.
- Succint and professional code written.
- Excellent indentation and commenting of code.
- **Impress us with the quality of your work.**

To achieve a very good (60-69%) submission:
- Satisfied over 70-89% of test cases
- Clear evidence that you have thought how to write this code effectively.
- Good indentation and comments, with some minor issues.

To achieve a good (50-59%) submission:
- Satisfies 50-69% of test cases.
- Code itself is functional albeit sub-optiaml and could be written more succinctly.
- Good indentation, commentary may be poor and lacking.

To achieve a satisfactory (40-49%) submission:
- Class implemented to specification.
- Class compiles with no errors.
- Code is functional..
- Questionable indentation and commentary.
- Passes less than 50% of the test samples.

To achieve a marginal fail (35-39%):
- Fails to adhere to the method signatures and class name given in document.
- Compiler errors persist.
- Poor coding style.

To achieve a fail (<35%):
- There must be some evidence of code having been attempted. No code means no submission!
- No evidence of structured coding style..