Instructions for C# Exercises:

• Implement the solution for Exercises #1 and #2 using C#

• The solution, should be \*compressed (zipped). The file must

o Contain a Visual Studio project file & solution file

o Compile and link

o Execute and create the expected output

o \*Before the project is compressed, remove any executable, object and debug files (usually generated by the program and stored in the debug or release folder within the project).

Exercise 1.

Write a command-line program that accepts two integers as arguments on the command line. The program counts backwards by ones from the first number until it reaches zero. While counting down, if the current count is evenly divisible by the second number, then the current count is written to the console window.

Example of positive use case:

Invoke program: C:\>BackwardsCounterProgram 25 5

Expected output: 25 20 15 10 5

Include robust error handling for the following negative use cases:

Sample inputs

- First number < 2 1 1

- Second number = 0 25 0

- Second number > first number 5 25

- First number not evenly divisible by second number 25 7

- Either number is not an integer 25 xyz

- Either number is negative -25 5

- First number must be < 1000 1010 10

Exercise 2.

Write a program in C#/.NET that contains the following:

An enumerated type, representing the possible resource types in the program, containing the following values: Hardware, Firmware, Software, and Other.

A class, entitled Resource, that contains two pieces of data. The first is an instance of the enumerated type above. The second is an integer that represents the number of units of the enumerated type specified in the first piece of data. The class should have a constructor or member function that accepts the enumerated type and the value of the units variable.

A class, entitled Requirements, that contains two pieces of data. The first is an instance of the enumerated type above. The second is an integer that represents the required number of units of the enumerated type specified in the first piece of data. The class should have a constructor or member function that accepts the enumerated type and the value of the units variable.

A class, entitled Provider, which contains a collection of zero to four instances of the Resource class. The class should have member functions that allows for addition and removal of Resources. No instance of the Resource class that is added to the Provider should be able to have the same value of the enumerated type as another instance currently present in the Provider.

A class, entitled, Demands, which contains a collection of zero to four instances of the Requirements class. The class should have member functions that allow for addition and removal of Requirements. No instance of the Requirements class that is added to the Demands should be able to have the same value of the enumerated type as another instance currently present in the Demands.

A method which takes an instance of a Provider, and an instance of a Demands, and returns how many whole multiples of the set of Requirements inside the Demands instance are present in the set of Resources inside the Provider instance.

One or more test methods which:

1. Create an instance of a Provider with different numbers and combinations of Resources;

2. Create an instance of a Demands with different numbers and combinations of Requirements;

3. Invoke the previous method using the Provider and Demands from (1) and (2) above;

4. Verify that the value returned from the method is correct.