LAB 2

Note: This lab has **two different C# solutions in the zip file**. Use the Debug Exercise solution for Tests 1 and 2. Use the Lab 2 solution for Tests 3-10.

Note: Prior to attempting to open the program, **you must first 'extract' the zip file**. Visual Studio will not correctly access a project that is in a zip file. You must use the extracted file/project to work on the project in Visual Studio

Test 1 – Debug a program

Open the Debug Exercise program (double click on the Debug Exercise.sln file). Correct all syntax errors (there should only be 5 – simple errors and will not require any new code – just correcting minor or typographical errors) in the Program.cs file so that it will run correctly. Do not remove any of the breakpoints. If they are inadvertently removed, they should be on lines 28, 30 and 60 (unless the program is re-formatted in the process of correcting syntax errors).

Test 2 – Use the Visual Studio Debugger

Open the Debug Questions document in the Debug Exercise project folder and answer the questions as the program executes. When you create the Debug Exercise zip file, ensure you include the Debug Questions document that includes you answers as part of the zip file.

NOTE: For Tests 3-10, you are required to use the appropriate Parse method for each conversion. The appropriate classes are listed in each Test's instruction (for Test3, it specifies to use the Int32 class – so Int32.Parse(input);

Beginning with Lab 2 – you will use the Submission class (Submission.cs) to respond to the questions/instructions for all Labs (do not edit any existing code in any other Lab files unless specifically told to). This does not preclude you from creating additional .cs files and classes.

Test 3 – Convert a string to int

```
static int Test3(string input)
```

Given a string, `input`, and using the `Int32` class, convert the string to an integer. Return the resulting integer.

Test 4 – Convert a string to sbyte

```
static sbyte Test4(string input)
```

Given a string, `input`, and using the `SByte` class, convert the string to a signed byte. Return the resulting signed byte

Test 5 – Convert a string to double

```
static double Test5(string input)
```

Given a string, `input`, and using the `Double` class, convert the string to a double. Return the resulting double

Test 6 – Convert a string to ushort

```
static ushort Test6(string input)
```

Given a string, `input`, and using the `UInt16` class, convert the string to a unsigned short. Return the resulting unsigned short

Test 7 – Convert a string to float

```
static float Test7(string input)
```

Given a string, `input`, and using the `Single` class, convert the string to a float. Return the resulting float

Test 8 – Convert a string to uint

```
static uint Test8(string input)
```

Given a string, `input`, and using the `UInt32` class, convert the string to an unsigned integer. Return the resulting unsigned integer

Test 9 – Convert a string to short

```
static short Test9(string input)
```

Given a string, `input`, and using the `Int16` class, convert the string to a short. Return the resulting short

Test 10 – Convert a string to ulong

```
static ulong Test10(string input)
```

Given a string, `input`, and using the `UInt64` class, convert the string to an unsigned long. Return the resulting unsigned long.

Additional notes/information:
When the instructions say 'given' it means that the test method is receiving the specified data as a parameter (the data inside the parentheses). You can use the variables directly in your solution. For example, Test3 has the 'header' line:

```
public static int Test3(string input)
```

- `public` and `static` are 'optional' modifiers for the header
- `int` is the return data (the 'type' of answer the method will provide)
- `Test3` is the name of the method
- `(string input)` is the parameter list. In this case there is a single parameter – it is a `string` and its name is `input` – it is the 'information' that is given (passed) to the method when it is called