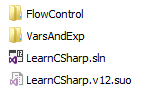
# C# Exercises: Variables, types, expressions, booleans flow control, functions

Download and follow the instructions in <https://github.com/viljoed/gis4x07/raw/master/GIS4x07_ExerciseSetup.docx> to create a Git repository for this exercise called gis4207-day08

Copy/paste the .gitignore file provided into the same folder that has the .git subfolder. The .ignore file ensures that assemblies (i.e. exe, dll) and other files/folders (e.g. .vs) that should not be under version control will be ignored.

1. In Visual Studio,
   1. Create a blank solution with the name LearnCSharp in the src folder
   2. Save the solution (File – Save)
   3. Close the solution (File – Close)
2. In Explorer, move the files from LearnCSharp up one level (into src) and then delete the LearnCSharp folder (i.e. the solution files should be in DavidV\src)
3. In Visual Studio, add 2console projects to the Solution called  
   1. VarsAndExp
   2. FlowControl
4. Your folder structure under src should look like the following:  
   
5. You may or may not see the .vs folder since it is hidden and its visibility depends on the folder options you have set. If this folder does not exist (e.g. you delete it accidently or on purpose), it will be created the next time you open the .sln file in Visual Studio. Files in this folder track things like which files were open in Visual Studio, which was the StartUp project, etc.

You can now open the LearnCSharp.sln and you should see the same content in the Solution Explorer. If not, stop and ask for help.

I would strongly recommend that when you get something working, commit your changes to the repository so that you can revert back to your previous working version if something goes wrong.

We will now add some other content to each of these projects. Switch your StartUp project to the one you are working on (right-click on the Project in the Solution Explorer and select “Set as StartUp Project”).

Every function you add should:

* Use the static keyword.
* Have one or more calls in Main to test whether or not they are working as expected.

## VarsAndExps

1. In the program class add a static function called Dms2Dd that returns a double and has four parameters. The first three are named deg, min, and sec and have the type int, int, and double respectively. The last is named bearing and is a string that contains the quadrant of the dms coordinate (i.e. N, S, E, W).
2. On the line above the Dms2Dd function, create a /// comment and add some descriptive text
3. In the Main function, create a variable called dd of the type double. Set this variable to the return value of a call to Dms2Dd with some values that will test whether or not the expression is giving expected results. Note as you are adding the call that the text you put in the /// comment is helping you with the usage of this function.
4. Add a call to Console.WriteLine that has the expected and actual values of dd.
5. Add pairs of lines for other possible combinations of parameters. The first line of the pair assigns dd to the return value of a call to Dms2Dd and the second line uses ConsoleWriteLine to show the expected and actual value. The parameter values should test the limits of the number ranges with valid and invalid values.
6. Here are the business rules for this function:
   1. The first three parameters must be positive numbers
   2. The deg parameter must be <=180 if bearing is E, W, e, or w
   3. The deg parameter must be <=90 if bearing is N, S, n, or s
   4. The min and sec parameters must both be < 60
   5. If the bearing parameter is S or W, then the returned value is negative
   6. If the bearing parameter is N or E, then the returned values is positive
   7. If any of the parameters are not valid, then return 9999
7. Use “if” for the first three parameters and a “switch” for the bearing parameter
8. Once you have something working, commit your code so you could revert back to this working state if you needed to.

## FlowControl

1. Add a static function called CountDownWithFor to Progarm.cs that has one integer parameter called fromValue but does not return anything to the calling statement. Use a for loop to Console.WriteLine values from the fromValue to 0 skipping the number 1.   
     
   HINT: Use the continue keyword in combination with an “if” to skip the number 1.
2. Add as static function called CountDownWithWhile that has the same signature and output as CountDownWithFor but use a while loop.
3. Add a static function called CountDownWithDo that has the same signature and output as the previous two functions but uses a do … while loop.
4. Add calls to both CountDownWithWhile and CountDownWithDo in Main.

# Scope

In the program class of FlowControl, add the following variables above the main function:

        public static string g\_publicVar = "Public";

        internal static string \_internalVar = "Internal";

        private static string \_privateVar = "Private";

Add the public keyword before the “class Program” in FlowControl.

In the VarsAndExp project, add a reference to the FlowControl project (right-click References and select Add Reference – Projects)

Add the following function to VarsAndExp

        static void scopeTest()

        {

            FlowControl.Program.g\_publicVar = "Public";

            FlowControl.Program.\_internalVar = "Internal";

            FlowControl.Program.\_privateVar = "Private";

        }

Add a call to ScopeTest in Main of VarsAndExp.

Attempt to build/run this project.

Comment out any problematic lines of code in VarsAndExp. Add a comment explaining why you had to comment it out.

Below are some questions for you to consider. Expect to see questions on a future test that will assess your understanding of this.

1. Do you understand what the problem is with these lines of code?
2. Why did you add the public keyword to class Program in the FlowControl project?
3. Why did you add a reference from VarsAndExp to FlowControl?

Save and commit all work to your repository.