Debugging in the Console App

Create a new Console App named DebuggerLastName. Time to copy some code to learn some debugging strategies.

Code

- Add the following code to the Main method, add your first and last name on the final comment line. Read the comment, it informs you of the intent of the code:

```
static void Main(string[] args)
{
    // Call a method that returns the user's first name
    // first letter capitalized only
    // and the last name all capitalized followed by rank
    string winner1 = MedalWinner("MEGAN", "rapinoE", "1");
    string winner2 = MedalWinner("USA", "Women's Soccer","1");
    string winner3 = MedalWinner("USA", "Women's Soccer", "1.5");

    Console.WriteLine("And the winner is ... {0:G}", winner1);
    Console.WriteLine("Expected: Megan RAPINOE Rank: 1");
    Console.WriteLine("And the winner is ... {0:G}", winner2);
    Console.WriteLine("Expected: Usa WOMEN'S SOCCER Rank: 1");
    Console.WriteLine("And the winner is ... {0:G}", winner3);
    Console.WriteLine("Expected: Usa WOMEN'S SOCCER Rank: 1.5");

}
```

- Add the method MedalWinner() above Main:

```
private static string MedalWinner(string fName, string lName, int rank)
{
    string result;
    result = fName + " " + lName;
    result = result + " Rank: " + Convert.ToInt32(rank).ToString();
    return result;
}
```

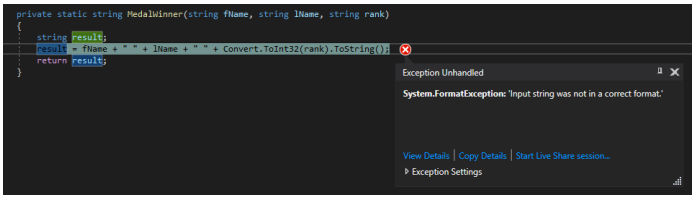
Notice that you cannot run the code in its current form. It's time to start Debugging!

Compile Error

Notice the underlines in the Method calls to MedalWinner with the "1", "1", and "1.5". That is because those are strings. The method parameter list indicates that rank is an integer (int rank). There are two possible fixes. Make them integers OR change the method signature from **int rank** to **string rank**. When you make this change, you are deciding what rank could be, should it be an int? Can be anything else for this use of MedalWinner?

For this lab assume the user is only sending the method strings. This is a fair assumption for input, especially in the Console App environment you are working with. In GUIs you often can limit the input type to numbers. You have seen this in Web applications for input like phone numbers and dates.

- Run the program once you have fixed the compile error. Notice the Unhandled Exception:



Break Point

- Set a break point in the method MedalWinner(), where the exception occurred.
- Take a screen shot of the break point, make sure the breakpoint and all the code is visible. (Screenshot #1)
- Run the code and take a screen shot with the code stopping at the error, make sure the code window is visible (Screenshot #2)

Debug

Coding involves a lot of debugging, finding errors and fixing them. **Do not change the code in the Main method.**

Exception Handling

Fix the unhandled exception by adding try/catch statements in the method MedalWinner()

- Add try to one line only, keep the return statement below the try/catch and the first two lines above the try catch.

```
try
{
    result = result + Convert.ToInt32(rank).ToString();
}
catch (            e)
{
    result = result + " Rank: undetermined";
}
```

- Be sure to catch the appropriate Exception, it is left blank for you to fill in just in front of the variable *e*. This is where you declare the data type for the exception your code is catching. You will see the type you need to catch in your Exception Unhandled.
- Run the code and notice it now compiles and runs without throwing an exception! Small victory, but do not take too long to celebrate, as you are not done yet.
- Notice the it does NOT follow the business logic that is that is visible in the Main method comment and the WriteLine of Excepted output. Recall you are not changing Main, so you need to change your code in MedalWinner() to match the expected output and business logic.
- First notice, 1.5 is an acceptable Rank for this lab. Add code convert to decimal instead of setting Rank to undetermined.

```
try
{
    result = result + Convert.ToInt32(rank).ToString();
}
catch (            e)
{
    result = result + Convert.ToDecimal(rank).ToString();
}
```

- Run the code. Notice it still compiles, it still does not throw an exception, and it prints 1.5 as expected.
- What does the variable e hold? You can test it by adding a debugging print statement.

```
Console.WriteLine(e.ToString());
```

- Remove the debugging print statement after you have examined the output it produces.
- It works until you add the following lines to your Main method

```
        string winner4 = MedalWinner("USA", "Men's Soccer", "GOLD");

        Console.WriteLine("And the winner is ... {0:G}", winner4);
        Console.WriteLine("Expected: Usa MEN'S SOCCER Rank: undetermined");
```

- Main now looks like

```
static void Main(string[] args)
{
    // Call a method that returns the user's first name
    // first letter capitalized only
    // and the last name all capitalized followed by rank
    //
    string winner1 = MedalWinner("MEGAN", "rapinoE", "1");
    string winner2 = MedalWinner("USA", "Women's Soccer", "1");
    string winner3 = MedalWinner("USA", "Women's Soccer", "1.5");
    string winner4 = MedalWinner("USA", "Men's Soccer", "GOLD");
    Console.WriteLine("And the winner is ... {0:G}", winner1);
    Console.WriteLine("Expected: Megan RAPINOE 1");
    Console.WriteLine("And the winner is ... {0:G}", winner2);
    Console.WriteLine("Expected: Usa WOMEN'S SOCCER 1");
    Console.WriteLine("And the winner is ... {0:G}", winner3);
    Console.WriteLine("Expected: Usa WOMEN'S SOCCER Rank: 1.5 ");
    Console.WriteLine("And the winner is ... {0:G}", winner4);
    Console.WriteLine("Expected: Usa MEN'S SOCCER Rank: undetermined");
}
```

- You need a nested try/catch for the decimal conversion, include it inside the catch. You cannot use the variable name *e* again for the second exception, here notice *e2* is used.

```
            try
            {
                result = result + Convert.ToDecimal(rank).ToString();
            }
            catch (              e2)
            {
                result = result + " Rank: undetermined";
            }
```

- Now the code in your method will contain nested try/catch. The variable name *e* is changed to *e1* for consistency and to note there are 2 exceptions.

```
    try
    {
        result = result + Convert.ToInt32(rank).ToString();
    }
    catch (               e1)
    {
        try
        {
            result = result + Convert.ToDecimal(rank).ToString();
        }
        catch (               e2)
        {
            result =
        }
    }
```

- What should you set result to here to denote it was not able to be determined?
- Almost done, run your code. Note there is still a business logic error.

Business Logic Error

The output is not as expected! Fix the business logic (first name format and last name format)

- The comment in Main tells you
    - `// user's first name first letter capitalized only`
    - `// and the last name all capitalized followed by rank`
- Use string processing you have learned to fix the user name format
- Take a screen shot of the code with ALL errors fixed, make sure the code window is visible with all code in the view (Screenshot #3)

Notice the types of errors you fixed. First, a compile error, next unhandled exceptions, and finally a business logic error. The end user informs the business logic (what the program should do, how the input should look, etc.) First you used the IDE itself, to notice you could not compile the code. Once you could compile the code, you found the unhandled exception in the IDE. Finally, you used the IDE's debugger to help fix exceptions and the business logic errors. Debugging is an important part of programming.

Submit all 3 Screenshots, either in a doc file, pdf file, or as 3 separate screenshots.