Validating Inputs

https://csci-1301.github.io/about#authors

January 17, 2023 (03:24:23 PM)

Contents

This lab serves multiple goals:

- To reinforce your understanding of `TryParse` statements,
- To help you understand the value returned by `TryParse`,
- To help you understand the difference between `int.TryParse` and `double.TryParse`,
- To make you familiar with the tools to validate user input,
- To have you validate different kinds of inputs from the user,
- (Optional) To manipulate user-input validation with classes.

1 Discovering TryParse's Behaviour

In your IDE, copy-and-paste the following:

```
Console.WriteLine("Enter… something!");
int answer;
bool valid = int.TryParse(Console.ReadLine(), out answer);
Console.WriteLine($"returns: {valid}\nvalue:{answer}");
```

For each input in the table below:

1. in returns column, write whether the TryParse operation succeeded (`true` or `false`).
2. in value column, write the obtained numeric value after `TryParse` operation.

The first few lines are given as examples, your task is to complete the rest of the table. You will need to update the program by replacing all the occurences of `int` with `double` to test if your answers were correct in the second half of the table.

	int.TryParse		double.TryParse	
Input	returns	value	returns	value
`"160519"`	`true`	160519	`true`	160519
`"9432.0"`	`false`	0	`true`	9432.0
`"nope"`	`false`	0	`false`	0
`"12,804"`				

int.TryParse	double.TryParse
`"+5102"`	
`"2+2"`	
`" -322 "`	
`"(72);"`	
`"000"`	
`"78 095"`	

Question: After completing the table, can you detect a pattern between "returns" and "value"?

2 Validating Inputs

For the following problems, always perform these steps:

1. ask the user for input,
2. check that the input is valid according to the specific problem,
3. perform the subsequent action.

If the provided input is not valid, request new input from the user until the user provides valid input. The beginning of the first and second problems are given to get you started.

1. Write a loop that displays: `Enter yes to quit:` then checks the user's input. Consider any of these variations to mean yes: "yes", "YES", "y", "Y". Once the user enters yes, exit the loop.

   Solution (sketch)

   ```
   Console.WriteLine("Enter yes to quit.");
   string answer;
   answer = Console.ReadLine();
   while (answer != "yes"){
       Console.WriteLine("Enter yes to quit.");
       answer = Console.ReadLine();
   }
   Console.WriteLine("You exit the program.");
   // Note that this program is not a complete solution: "YES", "y" or "Y"
   // do not make the program quit.
   ```

2. Ask the user to enter a positive integer, between 2 and 100 (including the boundary values 2 and 100). Make sure the value the user enters is between these bounds. Then compute the sum of integers starting from 1 up to the integer user entered, and finally display that sum. Here are examples:

   - if the user enters 5, compute: $1 + 2 + 3 + 4 + 5$, then display 15 on the screen
   - if the user enters 8, compute: $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8$, then display 36 on the screen

   Solution (sketch)

   ```
   Console.WriteLine("Enter a number between 2 and 100.");
   int answer;
   answer = int.Parse(Console.ReadLine());
   while (answer < 2){
       Console.WriteLine("That number is too small!");
       Console.WriteLine("Enter a number between 2 and 100.");
       answer = int.Parse(Console.ReadLine());
   }
   Console.WriteLine("You exit the program.");
   ```

```
// Note that this program is not a complete solution.
// Values greater than 100 are not rejected,
// And the final calculation is not done.
```

3. Do this next problem using the `decimal` type. Ask the user to enter any numbers (can be positive, negative, or zero). Ignore all non-numeric inputs, using `TryParse`. Choose an appropriate sentinel value to enable the user to indicate when they are done. Compute and display the average of all the numbers the user entered. If the user entered no numbers, display "You did not enter any numbers".

   Here is an example of the desired execution, where the user input is underlined, and hitting "enter" is represented by ↵:

```
Please enter a number, or "Done" to exit:
8↵
Please enter a number, or "Done" to exit:
2↵
Please enter a number, or "Done" to exit:
Hold on↵
Please enter a number, or "Done" to exit:
-5↵
Please enter a number, or "Done" to exit:
Done

The average of the numbers you entered is 1.66666666667.
```

3 Pushing Further (Optional)

This part is focused on input validation with classes. It requires to read a lengthy (but not very complicated) class implementation, and to improve it. It is *difficult*, and tailored to offer an interesting challenge. However, you should be able to complete such exercises by the end of the semester without too much difficulties.

Start by downloading the LoanCalculator[1] solution, which mixes classes and decision structures. Spend some time studying the implementation to understand *what* the program is doing and *how* it is doing it.

Next edit the `Program.cs` file of the `LoanCalculator` solution to add the following validation features:

1. Users entering a value other than `A`, `a`, `H`, `h`, `O` and `o` for the loan type will be asked again, and asked as long as they do not give a valid answer.
2. Users entering a credit score not between 300 and 850, or that is not an integer, will be asked again, and asked as long as they do not give a valid answer.
3. Users entering an amount needed or a down payment that is not a decimal, or is a negative decimal, will be asked again, and asked as long as they do not give a valid answer.
4. (Optional) Use the ToLower()[2] or ToUpper()[3] methods of the `char` class to make the program more readable – you will be able to greatly simplify the `if` statement that checks the loan type.
5. (Optional, hard) Write a method for the Loan class that takes a character as an argument, and returns the string describing the type of loan designed by that character. Then, use this method in the `ToString` and in the application program instead of doing it "by hand".

You can find a possible solution in this archive[4].

[1] labs/ValidatingInput/LoanCalculator.zip
[2] https://docs.microsoft.com/en-us/dotnet/api/system.char.tolower?view=netframework-4.7.2
[3] https://docs.microsoft.com/en-us/dotnet/api/system.char.toupper?view=netframework-4.7.2
[4] labs/ValidatingInput/Solution_LoanCalculator.zip