DEVELOPER NOTES

Approach:

I would like to note that I have developed this program in regards to SOLID principles. As it comes to design pattern my choice was Builder Pattern with a slight modification for validation purposes.

Program Structure:

After entering input, a validation is performed on a number of input parameters. When the input is of an invalid length an exception will be thrown. Otherwise, a DateManager object performs the actual logic based on the predefined requirements. Therefore, SetOfDates and DateBuilder objects are instantiated.

The SetOfDates object is capable of holding both of the expected dates. Additionally, a List<Date> data structure is chosen to incorporate scalability. All of the SetOfDates's methods have been developed for the sake of managing only two of the dates. The DateBuilder object is responsible for instantiating a Date object after performing validation, which is in turn, delegated to a GregorianDateValidator object deriving from an IDateValidator interface. A DateBuilder's property - IDateValidator ensures that actual implementation of validation can be easily performed by another class deriving from IDateValidator interface.

Actual data is being validated inside of GregorianDateValidator methods. The result is then returned to the DateBuilder object in a state of a DateValidation object. The DateBuilder object checks for the Boolean IsValid property and throws an exception when GregorianDateValidator does not flag the DateValidation object as correct.

Final Remarks:

The validation implementation of GregorianDateValidator is pretty straightforward. However, it is worth mentioning that all the date formats are being tested unless a single valid combination is found. I have manually tested each possible route of the user input combination and everything works as expected. All of the methods crucial for the program logic are commented out for your convenience.

Postscript:

I have received an answer to my previous question and it stated that I can incorporate .NET Framework functionality in terms of entering proper date format. However, my original approach was to test for the first valid arrangement of date components composed out of the given input. This idea seemed more challenging, thought-provoking and basically more fun to me. Consequently, I had to take into account a required, more complex structural design of the program. On top of that, DateTime object holds redundant data, which, according to my task, won't be needed at all. Considering all of the above, I decided to send you my initial version of the program. However, if you want to see my ability to properly use DateTime class with an application of CurrentCulture .NET Framework functionality, below are required alterations of my program.

IDateValidator.cs

Erasure of the those methods is required: IsInMonthRange(), IsInDayRange() and SetUpCorrectDate().

DateComponents.cs

Insertion of the following constructor is essential.

```
public DateComponents(DateTime dateTimeValue)
{
    Day = dateTimeValue.Day;
    Month = dateTimeValue.Month;
    Year = dateTimeValue.Year;
}
```

GregorianDateValidator.cs

Replace existing GenerateValidationResult() method with the one listed below. IsInMonthRange(), IsInDayRange(), IsLeapYear(), IsDateValid(), SetUpCorrectDate(), SplitInput(), ValidateSplittedStringLength(),methods are all redundant as well as DateComponents and DateComponentsOrder properties. Corresponding using directive is also needed - `using System.Threading;`

```
public DateValidation GenerateValidationResult(string input)

{
    // Assemble temporary object
    DateValidation validationResult = new DateValidation();

    try
    {
        // Exception will be thrown whenever conversion is not successful
        DateTime dateTimeValue = Convert.ToDateTime(input,
Thread.CurrentThread.CurrentCulture);

        // Assign proper values
        validationResult.DateComponents = new DateComponents(dateTimeValue);
        validationResult.IsValid = true;

        return validationResult;
    }
    catch (FormatException ex)
    {
        validationResult.IsValid = false;

        Console.WriteLine(ex.Message);
        Console.WriteLine("Terminating program.");
        Environment.Exit(ExitCodes.InvalidNumOfArgsExitCode);

        return validationResult;
    }
}
```

Kamil Nowak