Replacement Coursework 1 (Roman Numerals)

This coursework is worth 10%. It is about translating roman numerals into integers and also about validating roman numerals. The coursework is due on 2 February at 5pm. Make sure the files you submit can be processed by just calling `scala <<filename.scala>>`.

Important: Do not use any mutable data structures in your submission! They are not needed. This menas you cannot use `ListBuffer`s, for example. Do not use `return` in your code! It has a different meaning in Scala, than in Java. Do not use `var`! This declares a mutable variable. Make sure the functions you submit are defined on the "top-level" of Scala, not inside a class or object. Also note that the running time will be restricted to a maximum of 360 seconds on my laptop.

Disclaimer

It should be understood that the work you submit represents your own effort! You have not copied from anyone else. An exception is the Scala code I showed during the lectures or uploaded to KEATS, which you can freely use.

Part 1 (Translation)

Roman numerals are strings consisting of the letters *I*, *V*, *X*, *L*, *C*, *D*, and *M*. Such strings should be transformed into an internal representation using the datatypes `RomanDigit` and `RomanNumeral` (defined in `roman.scala`), and then from this internal representation converted into Integers.

(1) First write a polymorphic function that recursively transforms a list of options into an option of a list. For example, if you have the lists on the left-hand side, they should be transformed into the options on the right-hand side:

```
List(Some(1), Some(2), Some(3))  ⇒  Some(List(1, 2, 3))
List(Some(1), None, Some(3))     ⇒  None
List()                           ⇒  Some(List())
```

This means the function should produce `None` as soon as a `None` is inside the list. Otherwise it produces a list of all `Some`s. In case the list is empty, it produces `Some` of the empty list. [1 Mark]

(2) Write first a function that converts the characters *I*, *V*, *X*, *L*, *C*, *D*, and *M* into an option of a `RomanDigit`. If it is one of the roman digits, it should produce `Some`; otherwise `None`.

Next write a function that converts a string into a `RomanNumeral`. Again, this function should return an `Option`: If the string consists of I, V, X, L, C, D, and M only, then it produces `Some`; otherwise if there is any other character in the string, it should produce `None`. The empty string is just the empty `RomanNumeral`, that is the empty list of `RomanDigit`'s. You should use the function under Task (1) to produce the result. [2 Marks]

(3) Write a recursive function `RomanNumral2Int` that converts a `RomanNumeral` into an integer. You can assume the generated integer will be between 0 and 3999. The argument of the function is a list of roman digits. It should look how this list starts and then calculate what the corresponding integer is for this "start" and add it with the integer for the rest of the list. That means if the argument is of the form shown on the left-hand side, it should do the calculation on the right-hand side.

$$
\begin{array}{lcl}
M :: r & \Rightarrow & 1000 + \text{roman numeral of rest } r \\
C :: M :: r & \Rightarrow & 900 + \text{roman numeral of rest } r \\
D :: r & \Rightarrow & 500 + \text{roman numeral of rest } r \\
C :: D :: r & \Rightarrow & 400 + \text{roman numeral of rest } r \\
C :: r & \Rightarrow & 100 + \text{roman numeral of rest } r \\
X :: C :: r & \Rightarrow & 90 + \text{roman numeral of rest } r \\
L :: r & \Rightarrow & 50 + \text{roman numeral of rest } r \\
X :: L :: r & \Rightarrow & 40 + \text{roman numeral of rest } r \\
X :: r & \Rightarrow & 10 + \text{roman numeral of rest } r \\
I :: X :: r & \Rightarrow & 9 + \text{roman numeral of rest } r \\
V :: r & \Rightarrow & 5 + \text{roman numeral of rest } r \\
I :: V :: r & \Rightarrow & 4 + \text{roman numeral of rest } r \\
I :: r & \Rightarrow & 1 + \text{roman numeral of rest } r
\end{array}
$$

The empty list will be converted to integer 0. [1 Mark]

(4) Write a function that takes a string and if possible converts it into the internal representation. If successful, it then calculates the integer (an option of an integer) according to the function in (3). If this is not possible, then return `None`. [1 Mark]

(5) The file `roman.txt` contains a list of roman numerals. Read in these numerals, convert them into integers and then add them all up. The Scala function for reading a file is

```
Source.fromFile("filename")("ISO-8859-9")
```

Make sure you process the strings correctly by ignoring whitespaces where needed.

[1 Mark]

Part 2 (Validation)

As you can see the function under Task (3) can produce some unexpected results. For example for $XXCIII$ it produces 103. The reason for this unexpected result is that $XXCIII$ is actually not a valid roman number, neither is $IIII$ for 4 nor MIM for 1999. Although actual Romans were not so fussy about this,[1] but modern times declared that there are precise rules for what a valid roman number is, namely:

- Repeatable roman digits are I, X, C and M. The other ones are non-repeatable. Repeatable digits can be repeated upto 3 times in a number (for example MMM is OK); non-repeatable digits cannot be repeated at all (for example VV is excluded).
- If a smaller digits precedes a bigger digit, then I can precede V and C; X can preced L and C; and C can preced D and M. No other combination is permitted in this case.
- If a smaller digit precedes a bigger digit (for example IV), then the smaller number must be either the first digit in the number, or follow a digit which is at least 10 times its value. So VIV is excluded, because I follows V and $I * 10$ is bigger than V; but XIV is allowed, because I follows X and $I * 10$ is equal to X.
- Let us say two digits are called a *compound* roman digit when a smaller digit precedes a bigger digit (so IV, XL, CM for example). If a compound digit is followed by another digit, then this digit must be smaller than the first digit in the compound digit. For example IXI is excluded, but XLI is not.
- The empty roman numeral is valid.

The tasks in this part are as follows:

(6) Implement a recursive function `isValidNumeral` that takes a `RomanNumeral` as argument and produces true if **all** the rules above are satisfied, and otherwise false.

Hint: It might be more convenient to test when the rules fail and then return false; return true in all other cases. [2 Marks]

(7) Write a recursive function that converts an Integer into a `RomanNumeral`. You can assume the function will only be called for integers between 0 and 3999. [1 Mark]

(8) Write a function that reads a text file (for example `roman2.txt`) containing valid and invalid roman numerals. Convert all valid roman numerals into integers, add them up and produce the result as a `RomanNumeral` (using the function from (7)). [1 Mark]

[1] They happily used numbers like $XIIX$ or $IIXX$ for 18.