From Roman numerals to Arabic numbers

Joseph Wright *

2007/06/11

Abstract

The `unroman` package provides a number of functions to convert from Roman numerals to Arabic numbers. The set of functions is designed to cover a range of applications.

1 Introduction

TeX and the LaTeX kernel provide a number of functions for converting Arabic numbers to Roman numerals. However, there is a lack of functions to go the opposite way, back to Arabic numbers from a given Roman numeral. The `unroman` package seeks to address this, providing a flexible set of macros for a variety of applications.

2 Usage

2.1 Conversion from Roman numerals

`\unroman`
`\unromanhead`
`\unromantail`

The main aim of the package is to allow Roman numerals to be turned into Arabic numbers. This is achieved using the `\unroman{`⟨*numeral*⟩`}` macro, which takes a single argument ⟨*numeral*⟩. The case of ⟨*numeral*⟩ is unimportant, but the string must only contain valid Roman numeral characters (i, v, x, c, d, l, m). If the string contains any non-valid characters, `\unroman` returns the value 0, and writes a message to the log. The macros `\unromanhead{`⟨*numeral*⟩`}` and `\unromantail{`⟨*numeral*⟩`}` extract a Roman numeral from the beginning and end of a string, respectively. This is illustrated in Table 1.

`\unromanstore`
`\unromanheadstore`
`\unromantailstore`

Following the example of the `coolstr` package, versions of all three conversion macros are provided that do do produce any typeset output. Instead, they are designed to store the result in a LaTeX counter for further manipulation. So `\unromanstore{`⟨*numeral*⟩`}{`⟨*counter*⟩`}` will store the value of ⟨*numeral*⟩ in counter ⟨*counter*⟩. For example, `\unromanstore{mcmxcii}{mycounter}` yields `\the\value{mycounter}` of 1992.

*E-mail: joseph.wright@morningstar2.co.uk

LaTeX code	Output
`\unroman{MCMXCII}`	1992
`\unroman{ii--xi}`	0
`\unromanhead{ii--xi}`	2
`\unromantail{ii--xi}`	11

Table 1: Conversion examples

2.2 The length of a Roman numeral

`\romanheadlen` `\romantaillen` `\romanheadlenstore` `\romantaillenstore`

Using the core macros of `unroman`, it is easy to design methods for determining how many characters of a string constitute a valid Roman numeral. Obviously, if the entire string is expected to be a numeral, then the `\substr` macro from the `coolstr` package is applicable. The `unroman` package provides two functions to measure numerals at the end of strings, `\romanheadlen` and `\romantaillen`. For example, `\romantaillen{Figure~XI}` yields 2. Versions of both macros are provided which store the result in a counter, as `\romanheadlenstore` and `\romantaillenstore`

2.3 Reversing a string

`\reversestr`

The most efficient method for converting Roman numerals to Arabic numbers in LaTeX first requires the string to be reversed. For example, the numeral MCMXCII is first changed to IICXMCM before processing. As a result, `unroman` contains code to reverse a string. This is made available as a user command `\reversestr{`⟨*string*⟩`}`. A simple example is `\reversestr{Hello}`, yielding "olleH". This command is robust.

3 Implementation

3.1 Setup code

The first part of the package is concerned with basic identification and loading support packages. The only package needed is `coolstr`, which is used to provide a length-of-string function.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{unroman}%
  [2007/06/11 v1.0 From Roman numerals to Arabic numbers]
\RequirePackage{coolstr}
```

`urm@counta` `urm@countb` `urm@value` `urm@stringlength` `\ifurm@tailerror`

Various internal counters are needed by the package in order to function. These are declared here. A single new Boolean is also needed.

```
\newcounter{urm@counta}
\newcounter{urm@countb}
\newcounter{urm@value}
```

```
\newcounter{urm@stringlength}
\newif\ifurm@tailerror
```

3.2 Internal macros

`\urm@numtoint` The business-end macro of the entire package is the conversion of a single character to its value. Using any of the case-changing macros (*e.g.* `\lowercase`, `\MakeLowercase`, *etc.*) leads to problems. So the function tests for both upper- and lowercase letters. See also the LaTeX kernel function `\@slowromancap`, which takes a similar cautious approach to changing case.

```
\def\urm@numtoint#1{%
  \if#1i%
    1%
  \else%
    \if#1v%
      5%
    \else%
      \if#1x%
        10%
      \else%
        \if#1l%
          50%
        \else%
          \if#1c%
            100%
          \else%
            \if#1d%
              500%
            \else%
              \if#1m%
                1000%
              \else%
                \if#1I%
                  1%
                \else
                  \if#1V%
                    5%
                  \else%
                    \if#1X%
                      10%
                    \else%
                      \if#1L%
                        50%
                      \else%
                        \if#1C%
                          100%
                        \else%
                          \if#1D%
                            500%
```

```
                                \else%
                                  \if#1M%
                                    1000%
```

If the character hasn't been matched, then it is not a valid Roman numeral. The value -1 is used as an indicator of this.

```
                                  \else%
                                    -1%
  \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi%
}
```

`\urm@calc` The `\urm@calc` macro is the engine-room of the whole package.

```
\def\urm@calc#1#2\end{%
```

First, the current lead character is converted to its integer equivalent. This is stored in the first working counter

```
  \setcounter{urm@counta}{\urm@numtoint{#1}}%
```

The value returned by `\urm@numtoint` is examined to check for the warning value -1.

```
  \ifnum\the\value{urm@counta}>\z@%
```

The character returned is a valid Roman numeral. The next step is to compare to the value of the previous character. The value is then compared to that for the previous character (stored in `urm@countb`). Working from right to left, the current value is to be taken off the running total if it is less than the previous value. This is handled by sign inversion.

```
    \ifnum\the\value{urm@counta}<\the\value{urm@countb}%
      \setcounter{urm@counta}{-\the\value{urm@counta}}%
    \fi%
    \addtocounter{urm@value}{\the\value{urm@counta}}%
    \setcounter{urm@countb}{\the\value{urm@counta}}%
```

Having carried out the calculation, the string being processed is shortened by one character, ready for the next pass.

```
    \def\urm@string@a{#2}%
```

The case where `urm@counta` equals -1 value is now handled. Depending on which user function is in action, processing of the string may just stop. The alternative is to return a value of 0 and also provide suitable information in the run log.

```
  \else%
    \ifurm@tailerror%
      \PackageInfo{unroman}%
        {Invalid character in string --- returning value 0}%
      \setcounter{urm@value}{0}%
    \fi%
    \def\urm@string@a{}%
  \fi%
}
```

`\urm@reverse@string` The core of the string-reversing algorithm is picking off the first character of the string and using it to build a new string, in the opposite direction. Here, `\urm@string@b` is used to store the new string, while `\urm@string@c` holds the original information being "eaten" at each step.

```
\def\urm@reverse@string#1#2\end{%
  \edef\urm@string@b{#1\urm@string@b}%
  \def\urm@string@c{#2}%
}
```

`\urm@reversestring` The second internal function for reversing strings loops through each character of the supplied string. As this function is used for internal reasons, it doesn't print anything. This leads to the need for this to be an internal function.

```
\def\urm@reversestring#1{%
  \def\urm@string@c{#1}%
  \strlenstore{#1}{urm@stringlength}%
  \def\urm@string@b{}%
  \@whilenum\the\value{urm@stringlength}>\z@\do%
    {\expandafter\urm@reverse@string\urm@string@c\end%
     \strlenstore{\urm@string@c}{urm@stringlength}%
    }%
}
```

`\urm@roman@length` The core function for finding the length of a Roman numeral uses `\urm@numtoint` to determine whether characters in the string are valid Roman numerals. Once this stops, the string is truncated.

```
\def\urm@roman@length#1#2\end{%
  \setcounter{urm@counta}{\urm@numtoint{#1}}%
  \ifnum\the\value{urm@counta}>\z@%
    \addtocounter{urm@value}{1}%
    \def\urm@string@a{#2}%
  \else%
    \def\urm@string@a{}%
  \fi%
}
```

`\urm@romanlength` The counter for the length of a Roman numeral is set up here, and the same looping as used elsewhere is implemented.

```
\def\urm@romanlength#1{%
  \setcounter{urm@value}{0}%
  \strlenstore{#1}{urm@stringlength}%
  \def\urm@string@a{#1}%
  \@whilenum\the\value{urm@stringlength}>\z@\do%
    {\expandafter\urm@roman@length\urm@string@a\end%
     \strlenstore{\urm@string@a}{urm@stringlength}%
    }%
}
```

`\urm@make@head` `\urm@makehead` The functions to strip a Roman numeral at the start of a string are very similar

to those used to count how long such a string is. The only difference is the construction of a new string as the counting occurs.

```
\def\urm@make@head#1#2\end{%
  \setcounter{urm@counta}{\urm@numtoint{#1}}%
  \ifnum\the\value{urm@counta}>\z@%
    \edef\urm@string@b{\urm@string@b#1}%
    \def\urm@string@a{#2}%
  \else%
    \def\urm@string@a{}%
  \fi%
}
\def\urm@makehead#1{%
  \strlenstore{#1}{urm@stringlength}%
  \def\urm@string@a{#1}%
  \def\urm@string@b{}%
  \@whilenum\the\value{urm@stringlength}>\z@\do%
    {\expandafter\urm@make@head\urm@string@a\end%
     \strlenstore{\urm@string@a}{urm@stringlength}%
    }%
}
```

`\urm@unroman` The final internal macro is the loop to convert a Roman numeral into a number.

```
\newcommand*{\urm@unroman}[1]{%
```

The counters needed are initialised. The counter `urm@value` is used to hold the result of the conversion, whilst `urm@countb` is needed to track the value of the previously-converted character.

```
  \setcounter{urm@value}{0}%
  \setcounter{urm@countb}{0}%
```

The string is reversed, as this makes it much easier to calculate the value of the numeral.

```
  \urm@reversestring{#1}%
  \let\urm@string@a\urm@string@b%
```

The main loop then converts the value, iterating through the string one character at a time.

```
  \strlenstore{\urm@string@a}{urm@stringlength}%
  \@whilenum\the\value{urm@stringlength}>\z@\do%
    {\expandafter\urm@calc\urm@string@a\end%
     \strlenstore{\urm@string@a}{urm@stringlength}%
    }%
}
```

3.3 User space macros

`\reversestr` The string reversing macro simply calls the internal version of the function, then outputs the resulting string. Due to the interaction of the internal macros, this ends up as `\urm@string@b` rather than the more obvious `\urm@string@a`.

```
\DeclareRobustCommand*{\reversestr}[1]{%
  \urm@reversestring{#1}%
  \urm@string@b%
}
```

\unroman
\unromanstore
\unromantail
\unromantailstore
\unromanhead
\unromanheadstore
\unromanheadlen
\unromanheadlenstore
\unromantaillen
\unromantaillenstore

All of the user space functions to generate numbers come in two versions. The `store` versions are modelled on the `coolstr` package. All of these functions simple make appropriate calls to the internal functions, then deal correctly with the output.

```
\newcommand*{\unroman}[1]{%
  \urm@tailerrortrue%
  \urm@unroman{#1}%
  \the\value{urm@value}%
}
\newcommand*{\unromanstore}[2]{%
  \urm@tailerrortrue%
  \urm@unroman{#1}%
  \setcounter{#2}{\the\value{urm@value}}%
}
\newcommand*{\unromantail}[1]{%
  \urm@tailerrorfalse%
  \urm@unroman{#1}%
  \the\value{urm@value}%
}
\newcommand*{\unromantailstore}[2]{%
  \urm@tailerrorfalse%
  \urm@unroman{#1}%
  \setcounter{#2}{\the\value{urm@value}}%
}
\newcommand*{\unromanhead}[1]{%
  \urm@makehead{#1}%
  \expandafter\unroman\expandafter{\urm@string@b}%
}
\newcommand*{\unromanheadstore}[2]{%
  \urm@makehead{#1}%
  \expandafter\unromanstore\expandafter{\urm@string@b}{#2}%
}
\newcommand*{\romanheadlen}[1]{%
  \urm@romanlength{#1}%
  \the\value{urm@value}%
}
\newcommand*{\romanheadlenstore}[2]{%
  \urm@romanlength{#1}%
  \setcounter{#2}{\the\value{urm@value}}%
}
\newcommand*{\romantaillen}[1]{%
  \urm@reversestring{#1}%
  \expandafter\urm@romanlength\expandafter{\urm@string@b}%
  \the\value{urm@value}%
}
```

```
\newcommand*{\romantaillenstore}[2]{%
  \urm@reversestring{#1}%
  \expandafter\urm@romanlength\expandafter{\urm@string@b}%
  \setcounter{#2}{\the\value{urm@value}}%
}
```

Change History

v1.0

General: Initial public release 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.