Mark Cyzyk. Book Review: Andrew Watt's *Beginning Regular Expressions*, in *Technology Electronic Reviews* v13 (1), February 2006.

http://www.ala.org/ala/mgrps/divs/lita/ter/volume13no1.cfm#cyzyk

ter
Technology Electronic Reviews
Formerly *Telecommunications Electronic Reviews*

Volume 13, Number 1, February 2006

Technology Electronic Reviews (TER) is a publication of the Library and Information Technology Association.

REVIEW OF: Andrew Watt. (2005). *Beginning Regular Expressions*. Indianapolis, IN: Wiley Publishing Inc. (ISBN: 0-7645-7489-2).

by Mark Cyzyk

My programming career began with an attempt to compile a book-length bibliography of doctoral dissertations in philosophy. Such a large, textual work inevitably required substantial formatting and reformatting. In order to automate this process I learned all about the macro language supplied with WordPerfect 5.0 for DOS. It was simple, but powerful. As a programming novice at the time, the only routines I learned were simple search and replace functions. Running a very long macro on a very long piece of text on a very slow 8080 processor took a very long time, so I quickly learned that when searching and replacing certain patterns of strings in a large textual document one must be careful.

The machine, after all, is literal in a way that we humans typically are not. One of the great virtues of this book is that it points out the things that can go wrong when programmatically manipulating text via regular expressions and shows how those problems can be circumvented or solved.

But first, what is a regular expression? A "regular expression" is a pattern of characters in a text. It is "regular" insofar as it is identifiable and can therefore be used to match a pattern wherever it may occur throughout a text. Now, why the terminology here is denoted by the term "regular expression" instead of something more intuitive like "pattern matching" I don't know. But after all, we live in a technological world in which one of the key concepts of HTTP, the foundation of the World Wide Web, is spelled "referer". When confronted by the technical term "regular expression" I simply do an immediate internal translation to "pattern matching", and all is well.

This book consists of twenty-six chapters, the first ten illustrate the various aspects of regular expression technology, the remaining sixteen provide useful illustrations of how regular expressions are implemented in the top programming languages and applications. The core of the book is to be found in Chapters 3 through 9.

Chapter 3: Simple Regular Expressions

This chapter introduces and illustrates how to write a regular expression for use in matching patterns in text where the pattern to be matched consists of single characters, multiple characters in a string, optional characters, as well as using special "metacharacters", characters that have a special meaning within a regular expression to specify such things as match a class of characters; match any character; match within a certain range of characters; match a pattern zero or more times; match a pattern one or more times; and match a pattern a specified minimum and maximum number of times within a string of text. This is an introductory chapter to these topics which sets the stage for the chapters that follow.

Chapter 4: Metacharacters and Modifiers

A regular expression can be very simple, something for example like this: cat. This simple regular expression would match any sequence of a c followed by an a followed by a t in a string of text. So it would match that string as it appears in the words cat, cats, catalog, catastrophe, concatenation, etc. Metacharacters provide for more sophisticated pattern matching. Suppose for example you wanted to find all instances of cat or cut in a text. The period (.) metacharacter would allow you to do so: c.t. The period metacharacter matches any character. This chapter illustrates the use of the following metacharacters:

.	Matches any character
\w	Matches characters in the English alphabet, numeric characters, and the underscore
\W	Matches any character not matched by \w
\d	Matches a numeric digit
\D	Matches any character not matched by \d
\s	Matches a single space, a tab, or a newline character
\S	Matches any character not matched by \s
\t	Matches a single tab character

\n	Matches a single newline character escaped characters. Characters like the period and the backslash themselves must be "escaped" with a backslash: \. and \

Chapter 5: Character Classes

Character classes are convenient ways to match a range or entire classes of characters in a text. For instance:

[a-z]	Matches any single lowercase alphabetic character
[a-z]+	Matches any number of lowercase alphabetic characters
[A-Z]	Matches any single uppercase alphabetic character
[0-9]	Matches any numeric digit
[aeiou]	Matches any vowel

Parentheses can be used to group items:

(33|44)   Matches either a 33 or a 44 in a text

Use of quantifiers specify how many characters to match:

[aeiou]{2}   Matches any two consecutive vowels

Negated character classes can be specified with the metacharacter :

[^a-z]   Matches any single character that is not a lowercase alphabetic

Chapter 6: String, Line, and Word Boundaries

Metacharacter "anchors" are used to denote start and end of lines, and beginning and ends of words:

^	Matches the start of a line of text
$	Matches the end of a line of text
^$	Matches a blank line
\b	Matches either the beginning or the ending of a word
\<< />	Matches the beginning of a word
\>	Matches the ending of a word

Chapter 7: Parentheses in Regular Expressions

Just as one uses parentheses to group elements in a Boolean search query, so too are they used in regular expressions to group parts of the expression.

(Doctor|Dr|Dr\.)   Matches "Doctor" "Dr" or "Dr."

Watt briefly discusses, in this chapter, the notion of "capturing" parentheses, something that strikes me as being extraordinarily powerful. Essentially, programming languages and applications that allow for capturing parenthesis will match what is inside the parentheses and assign what is found there to a numbered variable name. The example Watt gives is:

(United) (States)

as used against the text:
The United States

In this case, the first parenthetical expression will match the string "United" in the text and will assign the value found there to a variable named "$1". The second parenthetical expression will match against the string "States" in the text and will assign the value found there to a variable named "$2". These two variables are now available for use by the programmer.

Chapter 8: Lookahead and Lookbehind
Another powerful feature of regular expressions is the ability to match a pattern based on a previously matched pattern or based on a pattern being followed by another pattern. This is called lookahead and lookbehind. There are several different flavors of these two features, but just as an example, here is what Watt uses to illustrate a "positive lookahead":

```
Star(?=Training)
```

Basically this regular expression says "Find the string 'Star', but only if it is immediately followed by the string 'Training'".

Lookahead and lookbehind, with their variants, surely are powerful programming constructs.

Chapter 9: Sensitivity and Specificity in Regular Expressions
The moral of this instructive chapter is: Sensitivity and specificity in a regular expression act in mutually opposing ways such that an increase in the sensitivity of an expression most often results in the decrease of specificity, and vice versa. So increasing either usually results in a tradeoff: The more sensitive your search, the more "false hits" are likely to result; the more specific your search, the more likely it will result in relevant items being excluded. Watt provides solid examples of this principle in action.

Chapters 11-26 discuss and illustrate the implementation of regular expressions in such applications, application programming platforms, and programming languages as: Microsoft Word; StarOffice/OpenOffice; findstr; PowerGREP; Microsoft Excel; SQL Server 2000; MySQL; Microsoft Access; JScript and Javascript; VBScript; Visual Basic .NET; C#; W3C XML Schema; Java; and Perl.

A chapter on regular expressions and Python would have been nice, but I suppose the line had to be drawn somewhere and an already very large book (742 pages) brought to an end. The other thing I would have liked to see is the provision of an Appendix briefly summarizing the syntax of regular expressions, a visual summary of the first several chapters -- sort of a Quick Lookup area of the book that could be easily accessible to the working programmer.

Overall, the book is informative, comprehensive, well-written, and is therefore easily readable. The examples are well thought out and clear. I think this book is ideal for someone starting out with regular expression programming and it also would have great value to the programmer who, for instance, needs to switch programming languages, from, say, Java or Perl to PHP.

In the end it's all about solving programming problems. Suppose, for instance, you are given a file of text consisting of what appears to be IP addresses, each on its own line, but you need to match just the strings that actually are valid IPs and weed out the rest. How to do this?

It's simple - just match on the following regular expression!:

```
^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])\.){3}(24[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])$
```

Mark Cyzyk, formerly Head of Information Technology in the Albert S. Cook Library at Towson University, is currently the Web Architect at Johns Hopkins University in Baltimore, Maryland.

About TER

Technology Electronic Reviews (TER) is an irregular electronic serial publication of the Library and Information Technology Association, a division of the American Library Association, 50 E. Huron St., Chicago, IL 60611. The primary function of TER is to provide reviews of and pointers to a variety of print and electronic resources about information technology. Resources include books, articles, serials, discussion lists, training materials, bibliographies, and other items of interest to librarians and information technology professionals. The topics covered may include, but are not limited to, networking technologies and standards; hardware and software; operating systems; databases; specific programming languages; management tools and utilities; technical project management; training and personnel issues; library perspectives; and research and development.

Opinions expressed in this publication are those of the writers and do not necessarily represent the viewpoints of LITA, ALA, or organizations involved in the storage and/or distribution of the publication.

TER is distributed electronically via Internet. There is no subscription fee. Currently it is available via World Wide Web (http://www.lita.org/ter/) and new-issue announcements are posted on the LITA-L electronic discussion list. To subscribe, send an email message to listproc@ala1.ala.org that says: subscribe LITA-L First-Name Last-Name. Other distribution arrangements may be made in the future.