東京家政学院筑波女子大学紀要第５集　209～228ページ　　2001年

＜研究ノート＞

正規表現について

－その１　どう読むか－

坂本　義行・江戸　浩幸

How to Master“ Regular Expression ”

SAKAMOTO Yoshiyuki and EDO Hiroyuki

概　要

コンピュータのプログラミング言語には、「正規表現」と呼ばれる表記法を備えたものがある。この表記法を用いることにより、一次元の配列からなる記号列を検索し、これを置換するのに便利な機能となる。それには厳密な表記規則（文法）があり、また、プログラミング固有の解釈あるいは処理がなされる。この正規表現を習得し、これを備えた最良の言語を用いることにより、前年度の紀要で報告した電子メール自動集計処理システムに応用することで、簡潔で理解し易いプログラムの作成と使い勝手の良いシステムの構築を目指す。今回はその第一段階として、「詳説正規表現」を読み、そこに記述されている基本的表記に対する厳密な解釈と動作状態を把握することに努めた経過を報告する。

キーワード：正規表現、文字列処理、メタ文字、リテラル文字、パターンマッチ

はじめに

コンピュータのプログラムには、『正規表現』†と呼ばれる強力なツール｛備えられた表記法を用いることにより動作する機能｝††がある。エディタ、ワープロ、設定用スクリプト、システム管理ツール等では、正規表現の機能が提供されていることが多い。Awk、

† 本書では、文章中の「」と正規表現を示す「」を区別していない。そこで、我々は文章中の「」記号を『』とした。

†† ｛ ｝：以後この記号で囲まれた部分は、著者等によって補足した説明を示す。

Elisp、Expect、Perl、Python、Tclといった言語には、この機能が組み込まれている。正規表現は、さらに、vi、Delphi、Emacs、Brief、Visual C++、Nisus Writerでも使われている。非常に日常的な機能といえる。

ここでは、「詳説正規表現」Jeffrey E.F. Friedl著、歌代和正監訳　オライリー・ジャパン発行の本を用いて、まず、その基本的な概念について学ぶことから始める。

その概念がたとえ簡単でも、初歩の人間にとって、その説明から正確に理解することは、容易ではない。また、正規表現自体は実際それほど難しいものではなくとも、その説明は複雑な場合が多い。

本書は、その全体を読み終えてからでなければ、リファレンスとして使っても意味がないと著者は述べている。したがって、われわれも章を追って、理解に努めたいと思う。全体は、導入、詳細解説、ツール別情報、付録の構成をとっており、導入部は、１章「正規表現とは」、２章「基本的な問題例」、３章「正規表現の機能と特性の概要」の３章から成り、これで一応基本知識を習得できると述べられている。そこで、今回はここまでを一区切りと考え、その説明を正しく理解するために、補足説明を加えながら読み進めた。

１．正規表現とは

正規表現は､汎用的なパターン記法でテキストの記述と解析を実現できる。

正規表現は、それを実装するツールの他の機能とともに用いることで、あらゆる種類のテキストおよびデータについて加える、削除する、抽出するなどの処理が可能である。本書では、生産性を向上させる正規表現の活用法を数多く紹介する。また、正規表現の考え方についても解説していく。PerlやPythonを使えば、重複単語の問題を解決する本格的なプログラムをほんの数行で実現することが可能である。

不要なテキストを無視して必要なテキストを見つける正規表現の書き方をまず学ぶ。

表記凡例

正規表現は、‘「’と‘」’ではさんだリテラルテキスト†で表記する	例：「this」
リテラルテキストは、‘.’のように表記する	例：‘this’
スペースは幕つきアンダースコアの記号を用いる	例：‘␣’
‘あ’と‘い’の間に４個のスペースがある場合	例：‘あ␣␣␣␣い’
タブ文字には ␉、	例：␉
改行文字改行は ␤、	例：␤
リテラルテキストや正規表現に下線を施し、強調を表記	例：「正規表現」
リテラルテキストおよび正規表現の中では、視覚的に明らかな省略記号を使った。例えば、[...]はブラケットセットの内容が特定されていないことを示し、また[. . .]はピリオドが３つ入ったブラケット††の組を示す。	

† リテラステキストとは、表記可能な文字列
†† ブランケットは‘[ ]’の角括弧で表記する。

1.1　文字列検索の例

egrepという一般的な検索ツールを用いて、単純なコマンドを打ち込み、各メッセージ中｛リテラル文字列‘From:’と‘Subject:’｝の行を取り出して表示させるのに、
「^(From|Subject):」を用いた。

1.2　言語としての正規表現

ファイル名の検索

“*.txt”というパターンが複数ファイルの選択に使えることを知っているはずだ。このようなファイル名のパターン†（ファイルグロブやワイルドカード††と呼ばれる）には、特別な意味を持つ文字がいくつかある。アスタリスクは『あらゆるものとマッチ』を意味し、疑問符は『任意の１文字とのマッチ』を意味する。「*.txt」の場合、まずあらゆるものとマッチする「*」で始め、最後に「.txt」というリテラルを書くと、『任意の名前で始まり、.txtで終わるファイルを選べ』というパターンができあがる。

強力なパターン言語及びそのパターン表現を一般的に『正規表現』と呼ぶ。

メタ文字とリテラル

本格的な正規表現は２種類の文字から成り立っている。この正規表現「*」は『メタ文字（meta character）』と呼ばれ、それ以外のものは全て『リテラル（literal）』と呼ばれるテキスト文字である。

正規表現自体が言語に当たり、リテラル文字が単語に、またメタ文字がその文法に当たると考えれば分かりやすいだろう。

筆者は、「^(From|Subject):」という表現を用い、メタ文字には下線を施した。

S!<emphasis> ([0-9]+ (\.[0-9]+){3})
<emphasis>!inet>$1<inet>!
という表現はじきにはっきりと分かるようになる。

本書の目標は、特定の問題の解決策を教えることではなく、どんな問題に直面してもそれを乗り越えられるよう、正規表現について考え方を伝えることにある。

テキストファイルの検索：egrep

テキスト検索は、最も単純な正規表現の使い方である。

egrepに正規表現と検索対象ファイルを与えると、その正規表現を各ファイルの各行と比較し、条件に一致した行だけを表示してくれる。

egrepは、最初のコマンドライン引数を正規表現として解釈し、残りの引数を検索対象ファイルとして解釈する。
｛脚注に、自分のシステム用にegrepを入手する方法は付録Aを参照と書かれている。この記述は、利用者には､大変便利だ｝

例えば､ファイル検索で「cat」を検索すると、連続する‘cat’といる３文字が含まれる行を全て見つける。Vacationにおける‘cat’という並びもれっきとしたマッチ対象になる。

重要なのは、正規表現による検索は単語単位ではないということだ。

1.3　egrepのメタ文字

行の先頭と末尾

もっとも理解しやすいメタ文字は、「^」（キャレット）と「$」（ドル記号）だろう。

チェック対象行の先頭と末尾を表す。

「^cat」は、行頭に存在する‘cat’のみと

---

† ここでのパターンとは、表記記号列を示す
†† ファイルグロブやワイルドカードは、広い領域で共通に使える記号列と解釈する

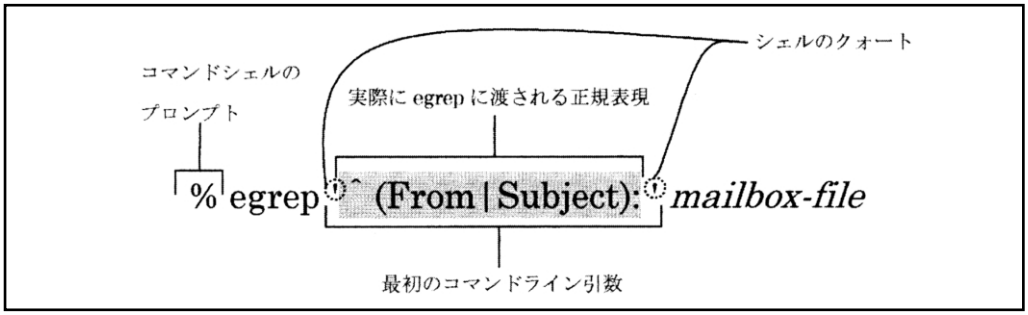

図1-1　**コマンドラインから**egrep**を呼び出す**

マッチする。同様に、「cat $」は、‘ scat ’で終わる行末のように、行末に存在するものだけを見つける。

すなわち、「^cat」は、行頭があり、その直後に‘ c ’があり、その直後に‘ a ’が続き、またその直後に‘ t ’が存在する場合にマッチする、と考える。

「^cat$」や「^$」はどう読めるか？

文字クラス

複数文字のいずれかとマッチする

例えば、‘ grey ’を検索したいが、‘ gray ’となっているものも対象とする。

「gr[ea]y」のように使用する。つまり、『‘ g ’を探し、次に‘ r ’を見つけ、その次に‘ e ’もしくは‘ a ’を探し、その後に‘ y ’を探せ』という意味になる。

ある語の先頭文字に大文字も許したい場合、例えば、「[Ss]mith」

HTMLヘッダを検索する場合、「<H[123456]」のように組み込む

文字クラスの内部にある‘ - ’（ダッシュ）という「文字クラス用メタ文字」は文字の領域を示す。

「[0-9]」や「[a-z]」は、それぞれ数字や小文字のアルファベットとマッチするクラスを表現する一般的な略記法である。複数の領域を指定してもかまはない。

つまり、「[0-9a-fA-F]」と書くこともできる。

領域とリテラル文字を組み合わせることもできる。例えば、「[0-9A-Z_!.?]」

ダッシュは文字クラスの中においてのみメタ文字となる点に注意されたい。

「^cat$」、「^$」および「^」の読み方

否定文字クラス

‘ ^ ’をクラスの中に入れると、そこに指定されたリテラルが否定される。

例えば、「^[1-6]」は１から６の数字以外の文字とマッチする。

‘ ^ ’はクラス外では位置指定文字であり、クラス内だとメタ文字になるが、これはクラスの開きブランケットの直後にある場合に限る。

例えば、% egrep ′q[^u]′ † word.listの実行（p11）

すなはち、否定文字クラスは『指定されていない文字とマッチせよ』という意味であり、『指定されているものとマッチしてはいけない』という意味ではない。

† egrepの中でシングルコート‘ ′ ’で囲まれた部分は、正規表現で「q[^u]」と解釈される

任意の文字とマッチさせる―ドット

「.」というメタ文字（通常ドットと呼ばれる）は、任意の文字とマッチする文字クラスを示す略記法だ。

ドットは文字クラスの中で用いられたら、メタ文字にならない。

例えば、「07[-./]04[-./]76」。
ただ、ドットはどんな文字でもマッチするという点を覚えておく必要がある。

検索対象テキストに関する知識と、『正規表現は常に厳密に書かなければならない』との要求の間でどうバランスを取るかは、繰り返し直面する重要な問題だ。

選択

複数の部分パターンのうちのいずれかとマッチする。

メタ文字の一つに「｜」がある。これは『または』という意味を持つ。これは、部分パターンを選択（alternative）と呼ぶ。

「gr[ea]y」は「grey|gray」とも、「gr (e|a) y」とも書ける。この場合の丸括弧もメタ文字である。

「^From|Subject|Date: 」
と「^ (From|Subject|Date): 」
を比較してみよう。

単語境界

単語の切れ目（語頭または語末）†とマッチする機能である。

メタ文字列として、「\ <」および「\ >」｛半角のバックスラッシュ｝というメタ文字列を使用する

例えば、「\ <cat \ >」は『catという単語を見つけよの』意味で、『単語の開始位置があり、その直後に‘ cat ’が続き、その直後に単語の末尾の位置があれば、それをマッチせよ』という意味になる。

「<」と「>」自体はメタ文字ではない。バックスラッシュと組み合わされた時に、はじめてこの文字列に特殊性が生じるのだ、『メタ文字列』と呼んだのはこうした理由による。

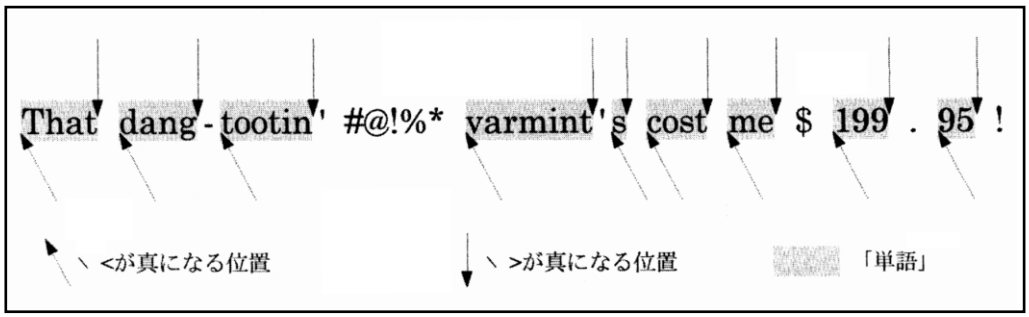

図1-2　「単語」の先頭と末尾の位置

† 「語の先頭と末尾」は「英数字文字列の先頭と末尾」を意味する

表1-1　これまで見てきたメタ文字の一覧

メタ文字	名　　称	意　　味
.	ドット	任意の1文字
[...]	文字クラス	列挙された任意の文字
[^...]	否定文字クラス	列挙されていない任意の文字
^	キャレット	行の先頭位置
$	ドル記号	行の末尾位置
\<	バックラッシュ、小なり記号	語の先頭位置†
\>	バックラッシュ、大なり記号	語の末尾位置†
|	または、縦線	隔てられた表現のうち、いずれかのものをマッチする。
(...)	丸括弧	|の対象領域を制限するのに用いる。また解説していない使い方もある。

これまでのまとめ

オプション要素

メタ文字である「？」（疑問符）は、それがオプション（任意）であることを意味している。

例：「coulou?r」は、「coulor」または「coulour」とマッチングする。

｛p17の上から13行目の‘?－’は説明がないが、オプションであるという意味に捉える｝

その他の制御文字：繰り返し

繰り返し制御文字（quantifiers）として、

「+」は、直前にある要素１個以上を意味する

「*」は、その要素がゼロを含む任意個あるを意味する

「<H[1-6]_*>」の意味

「<HR_+SIZE_*=_*14_*>」の意味

範囲指定の繰り返し

範囲指定繰り返し制御（interval quanifier）：

例：egrepの中へ「...{min,max}」として、「...{3,12}」と表記すれば、可能ならば12回までマッチ、最低でも３回なければならないと解釈する。

表1-2　繰り返し指定用メタ文字の一覧

	最低要求回数	最大繰り返し回数	意　　味
?	0回	1回	最低0回が要求される。1回まで許す（1回までが任意）
*	0回	無制限	最低0回が要求される。無制限に許す（何回でも任意）
+	0回	無制限	最低1回が要求される。それ以上も許す（1回以上が要求）

† egrepの中すべてのバージョンでサポートされているわけではない

大文字小文字の違いを無視する

文字の大小を一律に無視する

例：egrepで-iオプションを使う方法がある。

丸括弧と前方参照

丸括弧は、囲まれた部分パターンがマッチしたテキストを記憶することができる。前方参照とは、表現を各段階で対象となるテキストが厳密に分かっていなくても、すでに表現がマッチしたテキストと同じテキストが新たに出てきた場合、それをマッチできる正規表現の機能である。

エスケープ

全ての通常のメタ文字をバックスラッシュでエスケープするとは､例えば、
「\.」は文字クラスの内部を除き、「.」はメタ文字がエスケープされ、特殊な意味が失われ、通常のリテラル文字になる。

1.4　基礎を発展させる

言語の多様性

正規表現にも他の言語同様、「方言」や「訛り」が存在する。さらに、正規表現に多種多様な「ローカルバージョン」がある。

正規表現の目的

正規表現は何らかのテキストブロック（egrepの場合には行）とマッチするか、しないかのいずれかの結果となる。

いくつかの追加例

不要な時と場所でマッチさせないことが重要となる。

変数名

多くのプログラミング言語では、英数字とアンダースコアー†からなる識別子（変数名）を使うことができるが、数字で始めることができない場合が多い。

ダブルクォートで囲まれた文字列

ダブルクォートで囲まれた文字列は「"[^"]*"」という形で表される。

ドル金額

一つの表現は、「\$[0-9]+ (\.[0-9][0-9])?」である。

9:17am や 12:30pm のような時刻

一つの表現は、
「(1[012]|[1-9]:[0-5][0-9]_(am|pm)」

０時から23時までの24時間式時刻に応用すれば、以下の例のように表現すればよい。

「[01]?[0-9]|2[0-3]」

0	1	2	3	4	5	6	7	8	9
00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23						

「[01]?[4-9]|[012]?[0-3]」

0	1	2	3	4	5	6	7	8	9
00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23						

† アンダースコアーは、‘ _ ’を意味する

正規表現の用語

regex：正規表現をregexと本書では呼ぶ。

マッチ：正規表現が文字列の一部とマッチするという意味で使われている。

メタ文字：あるメタ文字がメタ文字となるという概念は、その正規表現が厳密にどこで用いられるかによって異なる。

特性

本書では、実装におけるこうした微妙な違いを総称して、「特性」という言葉で表現している。

部分パターン

「1-6」のようなものは、「H[1-6] _*」の部分パターンとは言えない。

文字

それを解釈するエンコーディング（符号化方式）によって異なる。

環境改善

異なる特性を持つ新しいツールに出会った場合、どういった違いやクセに注意すればよいかが分かる。

問題として、正規表現の利用法、正規表現の特徴、正規表現を実際にどう書くのかについては、プログラミング言語が違うと、表現が同じでも違った振る舞いをすることもままある。

筆者の目標は、正規表現を用いて問題を解決する方法を伝えることであり、これはとりもなおさず正規表現を組み立てるということである。

まとめ

表1-3　egrepのメタ文字のまとめ

メタ文字		マッチ対象
単一の文字とマッチする要素		
.	ドット	任意の文字１個をマッチ
[...]	文字クラス	指定された任意の文字１個をマッチ
[^...]	否定文字クラス	指定されていない任意の文字１個をマッチ
\ *char*	エスケープされた文字	*char*がメタ文字の場合、またエスケープとの組み合わせが特殊な意味を持たなければ、リテラル文字とマッチする。
付加することで「反復機能」が選られる要素：繰り返し文字		
?	疑問符	１個までオプションとして許す。
*	スター	いくつでもオプションとして許す。
+	プラス記号	かならず１個は要求するが、それ以上はオプション
(*min*,*max*)	範囲指定¶	*min*を最低条件として、*max*まで許す。
		位置とマッチする記号
^	キャレット	行頭位置とマッチ
$	ドル記号	行末位置とマッチ
\ <	単語境界¶	語頭位置とマッチ
\ >	単語境界¶	語末位置とマッチ
その他		
|	選択	この記号によって隔てられているいずれかの表現とマッチ
(...)	丸括弧	選択の対象範囲を制限する機能、繰り返し制御文字のためにグループ化を行う機能、および前方参照のための「格納」を行う機能。
\ 1,\ 2,...	前方参照	１番目、２番目の丸括弧の組に対応する、すでにマッチした文字列とマッチする。
	¶がついているものはegrepのすべてのバージョンでサポートされているわけではないことを示す。	

2．基本的な問題例

Perlの正規表現は、下記の３つを中心に構成されている。

例えば；

\b ([a-z]+) ((\s|<[^>]+>)+) (\1\b)

^ ([^\e]*\n)+

^

Perlは豊富なメタ文字セットを持っている。

2.1 例題について

Perlでは、egrepよりも格段に複雑な形で正規表現を使うことができる。

ここでは、ユーザ入力の確認、電子メールのヘッダの処理といった、いくつかの問題例を取り上げる。

２ページのファイルチェックの例をPerlで記述すると、

% perl -One ´print ″$ARGV\n″ if s/ResetSize//ig !=s/SetSize//ig´*

これはまだ理解できなくてかまわない。ただ答えの簡潔さに注目して欲しい。

Perlの簡単な解説

Perlは1980年代後半、Larry Wallが他の多くのスクリプト言語†からアイデアを集めて作った強力なスクリプト言語だ。テキスト処理††と正規表現に関するコンセプトの多くは、awkおよびsedと呼ばれる２つの言語に由来する。

Perlを自分のシステム用に入手する方法については、付録Aを参照のこと。

・単純な変数は、必ずドル記号で始まり、数値やテキストを格納することが出来る。
・コメントは＃で始まり、その行末まで続く。
・変数が、ダブルクォートで囲まれた文字列の中に置かれることがある。
“ $celsius C is $fahrenheit F \n ”という文字列では、各変数がその値と置き換えられる。
その出力は、
例えば、20 C is 68 F のように、

2.2 正規表現によるテキストのマッチ

変数$replyの中の文字列をチェックし、数字だけが含まれるかどうかを判別する例：

($reply =~m/^[0-9]+$/)の正規表現部分は、「^[0-9]+$」であり、それを囲むm/.../が、それに対する処理を指示している。mは『正規表現によるマッチを行え』という意味であり、スラッシュは正規表現自体を区切るために使われる。=~はm/.../を検索対象の文字列、つまりここでは変数$replyの内容に対応させる。すなわち、=~は、正規表現検索を検索対象の文字列と関係付ける。

「^[0-9]+$」と「[0-9]+」の違いは、前者は、$replyの内容には、数字のみが含まれているのに対し、後者の場合は、１文字でも数字が含まれていればよい。

もっと実際に即した例

プラス、マイナスのついた数値を許すためには、「[-+]?」を加える。

成功マッチの副作用

Perlは、マッチが終了した後に正規表現外部から参照できる変数も提供している。それは、$1, $2, $3などである。

---

† スクリプト言語とは、一次元の配列からなる文字列処理用プログラミング言語を指す

†† テキスト処理とは、一次元配列の記号列を処理する

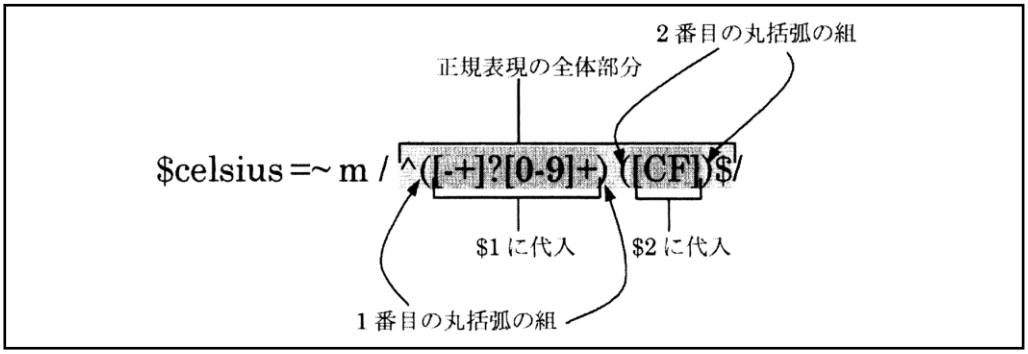

図2-1　格納型丸括弧

否定マッチ

　マッチしない場合、!~を用いる。

絡み合った正規表現

便利なメタ文字

「\n」（改行）、「\f」（改頁）、「\b」（バックスペース）等がある。

ちょっと寄り道－メタ文字の宝庫

　Perlのメタ文字列は独自のメタ文字を持っている。一部の文字列用メタ文字は、便利なことにそれに対応する正規表現用メタ文字と同じ形をしている。

　コマンド全体をコマンドシェルプロンプト†に入力すると、シェル自身が理解するメタ文字を認識する。

　シングルクォートは、引用符（シングルコート）で囲まれたテキスト内にある他のシェル用メタ文字を認識しないように命令するメタ文字となっている。

　文字クラスの中で使われる『サブ言語』は正規表現本体とは違う

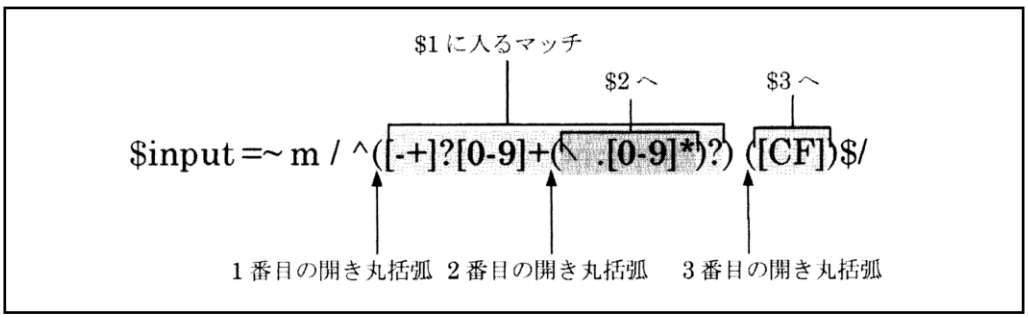

図2-2　丸括弧の入れ子構造

† コマンドシェルプロンプトとは、対話型処理システムで、ジョブ制御プログラムを呼ぶための記述。

\s は汎用の「空白文字」

「\ s」という略記法は空白文字に該当する文字クラス全体を表現する。これには（他にもあるが）スペース、タブ、改行､復帰が含まれる。

大文字、小文字を許す表現として、i修飾子を使うと、m/.../iで表記できる。

一休み

Perlの正規表現はegrepの正規表現とは異なる。より豊富なメタ文字を持っている。

Perlは、$variable =~ m/.../という構文†を使い、正規表現に照らして変数内の文字列をチェックすることができる。mがマッチを指示している。

Perlが提供している便利な略記法には次のものがある。

\t　タブ文字
\n　改行文字
\r　復帰文字
\ s　いずれかの種類の空白文字にマッチするクラス（スペース、タブ、改行、改頁など）
\S　「\s」以外のもの
\w　「[a-zA-z0-9_]」
\W　「\w」以外のもの」
\d　「[0-9]」つまり１桁の数字
\D　「\d」以外のもの。つまり「[0-9]」

修飾子/iを使うと、大小小文字を無視して検査する。マッチが成功すると、$1, $2, $3などの変数が参照できるようになる。

2.3　正規表現による整形

検索と置換について見ていく。

構文、$var =~ s/regex/replacement/

Perlでは、語頭と語尾を１つの文字列「\ b」で表せる。

「/g」は、グローバルマッチを表す。これは、１回のマッチが成功しても、すべての位置でのマッチを繰り返す。

自動編集

｛この節では、大量の編集作業が以下のコマンド一行で、行えたと説明。

％perl -p -i -e ‘ s/sysread/read/g ’ file

と書かれているが、説明不足で、自動編集の意味と正規表現の意味不明.｝

ちょっとしたメールユーティリティ

電子メールのファイルから、元のメッセージを引用して、自分の返答を必要な部分に移す方法として、元のメッセージのヘッダーから不要な行を削除して、自分の返信用ヘッダーを用意する。

作業は、次の３段階に分かれる。

・メッセージヘッダーからデータを抜き出す。
・返信用ヘッダーを出力する
・メッセージに ’ |>_ ’ をつけて出力する

データをプログラムに読み込む方法として、“ <> ”とい 、演算子を使う。

表題を取り出すには、

```
if ($line =~ m/^Subject: (.*)/) {
    $subject = $1;
}
```

‘ From: ’で始まる行を探すには、

「^From:|_ (\S+)_\ (([^()]*) ?)」

となる。

ここで、

「\ S」は空白文字ではないもの全てとマッチする。すなわち､最初の空白文字までとマッチする。

「\ (...\ )」は丸括弧とその括弧内の文字列とマッチする。

† 構文とは、表記形式を指す

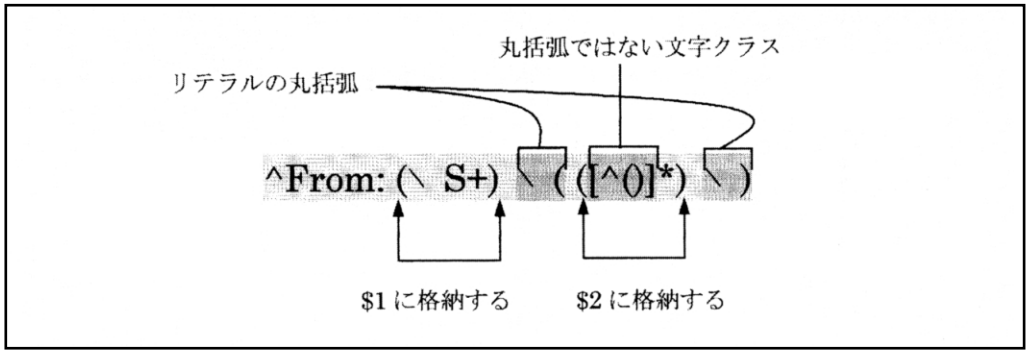

図2-3　入れ子になった丸括弧、$1 と $2

実際的な問題と実際的な解決策

Perlのdefined関数は、その変数が値を持っているかどうかを示し、die関数はエラーメッセージを出してプログラムを終了する。

実際の現実

電子メールは異なる多くのプログラムによって生成され、それぞれ独自の基準によっている。

３．正規表現の機能と特性の概要

正規表現の置き方、使い方、そして振る舞い方は、ツールによって途方もないほど異なる。

本章の目的

- 正規表現と、それを実現するツール の全体像を鳥瞰することにある。
- 興味深くまた価値ある洞察を得ることができる。
- 異なるツールで同じ問題をどのように解決できるかを熟知することは重要なことだ。
- 洞察力と広い視野を養っておくことが重要になるからだ。

3.1　正規表現の概要

grep以前の時代

1940年代に、Stephan Kleeneという数学者が、自ら「正則集合（regular sets)」と名付けた代数を使って、このモデルの形式を記述したことから正則表現が生まれた。

grepでサポートされていた正規表現は、きわめて限定的なものだった。

3.2　見かけ上の特徴

｛表の内容の読み方について、質問項目が挙げられているが、答えはない。｝

マッチの実行方式（あるいは少なくともそう実行されているように見える方式）に関する意味体系の違いは、きわめて重要な問題であるにもかかわらず、他の解説書ではしばしば見落とされている。

POSIX

Portable Operating System Interfaceの略であるPOSIXは、OS間での互換性を実現する標準規格である。

POSIX ロケール

ロケールという概念は､使用しているエンコーディングにおける文字の解釈などを記述する各種設定である。ロケールはプログラムの国際共通化を意図したものである。

POSIX 照合シーケンス

特定の文字または文字セットを、ソートなどの目的でどう扱うかを記述する照合シーケンス (collating sequence)というものを、ロケールによって定義することができる。

3.3　正規表現の扱いと注意

正規表現の意味と使い方をシステムに正確に伝えるために、より複雑なパッケージが必要になる。プログラムが正規表現をどう使うかを部分的に駆け足で見ていく。

正規表現を認識する

Perlでは、正規表現検索を指示するには、正規表現の周りにm/.../を使い、=~を使ってそれを検索対象テキストと関係付ける。

マッチしたテキストの処理

検索・置換コマンドの場合、置換文字列は正規表現ではない。

例：$var=~s/[0-9]+/<CODE>$&<\/CODE>/g

ここで、$&は、Perlの変数で、前回の正規表現マッチ（コマンドの最初の部分にある「[0-9]+」）でマッチしたテキストを示す。別のコマンドデリミタを使うことも出来る。

他のツール例

awk

正規表現による置換には、sub (...)という関数をもつ。

Tcl

正規表現も置換文字列にも、スラッシュや区切りスペースを除く区切り記号（デリミタ）は不要である。

GNU Emacs

正規表現を扱う数多くの関数を提供している。

表3-1で示すように、Emacsの正規表現の特徴として、バックスラッシュが大変多くなる。バックスラッシュは文字列用のメタ文字なので、‘ \ ’を正規表現の中で使うたびに（ \\ ’を使って）これをエスケープする必要がある。

表3-1　一般的なツール数種の特徴を（きわめて）大まかにまとめた一覧

機能	現代版 grep	現代版 egrep	awk	GNU Emacs バージョン19	Perl	Tcl	Vi
*, ^, $, [...]	✓	✓	✓	✓	✓	✓	✓
? + |	\?\+|	? + |	? + |	? + |	? + |	? + |	\?\+ ●
グループ化	\(...\)	(...)	(...)	\(...\)	(...)	(...)	\(...\)
単語境界	●	\<\>	●	\<\> \b, \B	\b, \B	●	\<\>
\w, \W	●	✓	✓	✓	✓	●	●
前方参照	✓	●	●	✓	✓	●	●

表3-2　POSIXによる正規表現の特性の概要

正規表現の機能	BRE	ERE
dot, ^, $, [...], [^...]	✓	✓
*, +, ?, {min, max}	*, ●, ●, {min, max}	*, +, ?, {min, max}
グループ化	\(... \)	(...)
くり返し制御文字が丸括弧に適用できる	✓	✓
前方参照	\１から\９まで	
選択	●	✓

Python

Pythonはこれまで取り上げてきたものとは異なり、野心的なオブジェクト指向型のスクリプト言語だ。

例　regex.set_syntax (RE_NO_BK_PARENS | RE_NO_BK_VBAR)

この２つの要素は、無修飾の丸括弧をグループ化用として、また無修飾のバー記号‘ _ ’を選択用として解釈させたいということを表している。

Pythonでは文字ケース†無視のマッチを許すのに面白い方法を用いている。

『増殖文字』††を無視させるように指定できる。¿を疑問符として、!を感嘆符として、また¢ ¤ £\などをすべて＄としてマッチさせることも可能だ。

正規表現の扱い方と注意点：まとめ

正規表現が単独で使われることはなく、枠組みを提供するユーティリティとともに用いられるため、しばしばそのツールが持つ、正規表現とは別次元の機能と関係してくることである。

3.4　エンジンとクロームフィニッシュ†††

正規表現に当てはめると、そのツールの提供する特性は、異なる２つの部分から構成される。１つの構成部分については、本章でこの後に取り上げるが、もう片方については次章で解説する。

クロームメッキと概観†††

正規表現の特性で一番はっきりと現われる違いは、メタ文字がどうサポートされているかである。

エンジンとドライバー†††

メタ文字の意味の実装と、それを組み合わせてさらに大きな表現を構築するセマンティクス††††は、きわめて重要な問題である。

・厳密にマッチする対象
・マッチのスピード
・マッチの後に使えるようになる情報（例えばPerlの$1およびその仲間）

---

† 文字ケースとは、複数の文字を同一の文字と解釈する
†† 一つ一つのバイト(文字)を同一の文字と解釈するように指示するために記述を与える
††† 本書では、自動車を例に表題が付けられているが、これは専門家以外では理解不可能な言葉や機能であり、正規表現での対応が認識できない
†††† 実際の処理動作を決定するという意味と捉える

3.5 一般的なメタ文字

ここでの概要は、ツールの違いを超えて共有できる要素や概念を含んでいる。

文字の略記法

多くのユーティリティでは、特定の装置以外では入力や視覚表示の難しい制御文字を表現するメタ文字が提供されている。

\a 警告音（この文字を「印字」するとベルが鳴る）通常は007（８進表記）のASCII<BEL>文字に対応する。

\b バックスペース 通常は010（8進）のASCII<BS>文字に対応する（後述するが、「\b」は多くの場合、単語境界のメタ文字として使われることも多い)。

\e エスケープ文字 通常は033（８進）のASCII<ESC>文字に対応する。

\f フォームフィード 通常は014（８進）のASCII<FF>文字に対応する。

\n 改行 大半の環境（UNIXやDOS/Windows）では、通常012（８進）のASCII<LF>文字に対応する。MacOSでは通常015（８進）のASCII<CR>文字に対応する。

\r 復帰 通常はASCII<CR>文字に対応する。MacOSでは普通ASCII<LF>文字に対応する。

\t （水平）タブ 通常は011（８進）のASCII<HT>文字に対応する。

\v 垂直タブ 通常は013（８進）のASCII<VT>文字に対応する。

機種依存性

大部分のツールにおいて、制御文字用略記法の多くは機種依存、コンパイラ依存である。すなわち、OS依存と考えることができる。

８進エスケープ －\数字

実装によっては、３桁の８進数エスケープを使って、特定の値を持ったバイト表現が出来るようになっている。８進エスケープは打ち込むことが難しい文字を表現内に入れるのに便利である。

「９は８進数か？」－ およびその他の変則的問題

リテラル‘９’と解釈する場合もあれば、９を（\11と同じ値の）８進数として解釈する場合もある。

16進エスケープ －\x数字

多くのユーティリティでは\xを使って16進値が入力できる。

正規表現を表す文字列

Emacs, Tcl, Python等のツールでは、正規表現オペランド†は通常文字列として渡される。つまりその言語の通常の文字列処理が先に行われてから、その結果が正規表現エンジンに渡されて正規表現として処理されるのである。表3-3で✓の付いた要素をサポートしているのは、正規表現エンジンではなく、この文字列処理なのだ††。

---

† 正規表現オペランドとは、関数における引数の役割を担うのがオペランドであり、正規表現内のリテラルと解釈する

†† この意味は、正規表現エンジンと文字処理とは独立に、入力処理においてリテラルの解釈処理を指すものと思われる。

バックスペースの「\ b」と単語アンカー† の「\ b」

PythonとEmacsは\bを２通りに解釈する。

「エスケープが落とされる」Emacs流の文字列処理

正規表現まで送り届けたいバックスラッシュは、それぞれエスケープしなければならない。

「エスケープをそのまま通す」Python 流の文字処理

Pythonの文字列は、認識できないバックスラッシュエスケープに対して反対の方法をとる。つまり手を加えずにそのまま通すのである。

クラスの略記法、ドット、文字クラス

\ d　数字　おおむね「[0-9]」と同じ

\ D　非数字　おおむね「[^0-9]」と同じ

\ w　単語構成文字

\ W　非単語文字　おおむね「\ w」を「＾...」で反対の意味にしたもの

\ s　空白文字　おおくの場合「[_\f\n\r\t\v]」と同じ

\ S　非空白文字　おおむね「\ s」を「[^...]」で反対の意味にしたもの

Emacs シンタックスクラス††

GNU　Emacsとその仲間は、「\ s」を用いて特殊な「シンタックスクラス」を参照する。

表3-3　数種のユーティリティと提供されているメタ文字の略記法

	（単語境界）\b	（バックスペース）\b	（警告音）\a	（ASCIIエスケープ文字）\e	（フォームフィード）\f	（改行）\n	（復帰）\r	（タブ）\t	\v	\8進	\16進	\d,\D	\w,\W	\s,\S	POSIX [:...:]
プログラム		**文字用略記法**										**クラス用略記法**			
GNU awkバージョン3.0.0	\ y	✓c	✓c	●	✓c	✓c	✓c	✓c	✓c	✓c▲	✓c▲	●	✓	●	✓c
GNU awkバージョン2.05	●	●	●	●	●	✓	●	●	●	●	●	●	✓	●	✓c
Perlバージョン5.003	✓	✓c	✓c	✓c	✓c	✓c	✓c	✓c	●	✓c	✓c	✓c	✓c	✓c	●
Tclバージョン7.5	●	✔	✔	●	✔	✔	✔	✔	✔	✔	✔	●	●	●	●
GNU emacsバージョン19.33	✓	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	●	✓	⇳	⇳
Pythonバージョン1.4bl	✓	✔	✔	●	✔	✔	✔	✔	✔	✔	✔	●	✓	●	●
Flexバージョン2.5.1	●	✓c	✓c	●	✓c	✓c	✓c	✓c	✓c	✓c	✓c	●	●	●	✓c
GNU egrepバージョン2.0	✓	●	●	●	●	●	●	●	●	●	●	●	✓	●	●

✓サポートあり　✔疑似サポート　✓cクラス内で有効

✓▲テキスト参照　●サポートなし　⇳サポートはないが、それに類似したものがある。

† 単語アンカーとは単語の語末を意味する

†† シンタックスクラスとは、メタ文字のような特殊機能を備えさせると解釈する

ほとんどの任意の文字－ドット

一部のツールでは、ドットを、すべての文字とマッチする文字クラスの略記法として使っている。行単位の処理では、ドットは改行を除く任意の文字とマッチするものになった。

文字クラス－[...]と[^...]

文字クラス内でしか認識されないメタ文字には、

・先頭のキャレット（否定クラスを表す）
・閉じたブランケット（クラスを終了する）
・領域演算子としてダッシュ（0123456789は0-9と略記できる）

ドット対否定文字クラス

「[^"]」といった否定クラスは改行とマッチする点に注意しよう。

POSIXのブラケット表現

通常文字クラスを呼ばれているものをPOSIX標準ではブラケットと呼んでいる。

POSIXブラケット表現の「文字クラス」

特殊メタシーケンスの一つで、[:lower:]はa-zに相当する。

POSIXブラケット表現における「相当文字」

同一とみなされる特定の文字を相当文字（character equivalents）と定義されている。

POSIXブラケット表現の「照合シーケンス」

ローケルでは照合シーケンスによって、ソートなどの際に特定の文字または文字セットをどう扱うかを指定できる。

位置指定

行頭や文字列の先頭位置を指定する－キャレット

edやgrepでは正規表現が調べるテキストは常に行単位だった。しかし、複数行にわたる文字列に正規表現を適用する場合には関係してくる。

キャレットは正規表現のどこでメタ文字になるのか？

文字クラスの外では、‘ ^ ’は位置指定用のメタ文字として解釈されたり、リテラルのキャレットとして解釈される。

行末用および文字列末尾用アンカー－ドル記号

単語境界－\ <...\ >と\ b\ B

どのツールも「単語用文字」の構成要素についてそれぞれ独自の定義を持っている。

グループ化と参照

前方参照がサポートされている場合、丸括弧に収められている部分パターンによってマッチしたテキストを参照する。

繰り返し制御文字

繰り返し制御文字（スター、プラス記号、疑問符、範囲指定などのように、修飾対象の出現回数に影響を与えるメタ文字）

範囲指定－｛min,max｝と\{min,max\}

範囲指定は、回数指定繰り返し制御文字と見なす。「x(0,0)」という表現は無意味である。

選択

選択は、複数の部分パターンのうちのいずれかを特定の位置でマッチさせるものである。

3.5 上級者編への指針

・ツールのマッチエンジンはそれぞれ異なる方式で実現されている。
・処理全体にどのくらいの時間がかかるのか

表3-4　文字列／行アンカーおよび改行に関連した問題

問題点	lex	Tcl	sed	awk	Perl	Python	Emacs
^が文字列全体の先頭とマッチする	✓	✓	✓	✓	✓	✓	✓
^が改行の後でマッチする（例えば、埋め込まれた論理行の先頭）	✓	●	●	●	●	✓	✓
$が文字列全体の先頭とマッチする	●	✓	✓	✓	✓	✓	✓
$が行末の改行の前でマッチする	✔	●	●	●	✓	✔	✔
$がすべての改行の前でマッチする（つまり、埋め込まれた倫理行の末尾）	✓	●	●	●	●	✓	✓
ドッドが改行とマッチする	●	✓	✓	✓	●	●	●
否定文字クラスが改行とマッチする	✓	✓	✓	✓	✓	✓	✓
注意：✔は「どこの改行の前でもマッチする機能の単なる副作用」である。 Perlのドットおよびアンカーの意味は変更が可能である。「文字列アンカー」を参照							

が異なる。
・知識とエンジンを最大限に活用する技法を考え出す。
・４章と５章は、本書の核心、すなわち神髄である。

ツール別情報

Perlが提供する、豊かで表現力ある正規表現インターフェースには、探求すべき点が数多くある。その最大の焦点は、Perlの持つ豊富な正規表現機能を理解・活用することにある。

あとがき

正規表現の基本的な表記と代表的なプログラミング言語上で、どのような動作を行うかについて学んできた。しかしプログラミング言語だけでなく、OSによる相違、あるいは装置（プラットフォーム）によっても、表記の違いがあり、また動作の違いもあることを学んだ。

正規表現を用いて、何らかの処理に利用するためには、実際のプログラムを書く際に、具体的な表記法ならびにプラットフォーム上での実際の動作を確認する必要がある。

今後は、本書による基礎的な知識の獲得と同時に、多くの強力な機能を備えていると言われるPerlについて、表記と動作の最適化を知ることによって、別稿で述べる「電子メール自動集計処理システム」の開発に応用していくつもりである。

参考文献

１）Jeffrey E. F. Friedl著、歌代和正監訳、1999、オライリー・ジャパン発行

２）江戸浩幸・坂本義行、「電子メール自動集計処理システムーⅠ」、東京家政学院筑波女子大学紀要、第４集、2000.

３）江戸浩幸・坂本義行、「電子メール自動集計処理システムーⅡ」、東京家政学院筑波女子大学紀要、第５集、200１.（掲載予定）