連載： eコンテンツのためのデータ加工 第4回

長谷川秀記¹

著者抄録：Webサイトの構築や広報システムの電子化などにもない各種のOAデータをeコンテンツ用のデータに加工する技術が必要とされる。この連載はデータ入手からテキスト化、データ・チェック、文字の問題、タグ・データ加工までの各段階について、基本知識と技術の実務を解説する。今回は「4. 取り出されたテキストデータの注意点とクリーンアップ」を開始するにあたって今後使用する「文字列加工のための基礎知識と道具」について概説する。

キーワード：データクリーンアップ、タグ付けルール、入れ子構造、メタ文字、エスケープ文字、テキスト・エディタ、ワープロ、マクロ、キーボード・マクロ、Perl、スクリプト言語、Sed、Grep、タグジャンプ、正規表現、置換

Series: Data processing for e-contents No. 4

HASEGAWA Hideki¹

Author Abstract: With building Web sites more, or with more PR systems being digitized ones are urged to know processing a diversity of OA data for e-contents. This series brief basic knowledge and practices in the course starting from data acquisition to data processing, that is, data acquisition, making texts of those data checking, characters, tagging and data processing. This article deals with the first part of chapter 4: Points attended in extracted text data, and clean up, that is, basic knowledge and tools for processing of character strings.

Key words: data clean up, tagging rule, nested structure, meta character, escape character, text editor, word processing, macro, keyboard macro, Perl, script language, Sed, Grep, tag jump, regular expression, substitution

原稿受理 (2002-02-26)

(情報管理 45 (1): 42-57)

4. 取り出されたテキストデータの注意 点とクリーンアップ

前回まではOAアプリケーションや印刷用データからテキスト形式のデータを取り出す過程を解説した。これでテキストデータという良質の材料が手に入り、いよいよこれらのデータを加工してeコンテンツを作成していくわけだ。しかしコンテンツ用の加工に入る前に必ず行わなくてはならないことが、手に入ったデータをチェックし、クリーンなデータとする過程である。

この過程をしっかりと行わないと次のタグ加工でさまざまなエラーに悩まされることになる。この章ではデータのチェックと修正について解説するが、この過程以降ではいろいろな文字列処理の道具と機能を使うことになる。そこでまず文字列加工のための道具と基礎知識について解説し、その後にデータのチェックとその修正方法を解説する。

4.1. 文字列加工のための基礎知識と道具

ここではテキスト形式のデータのチェックや加工に必要な基礎知識とツールとについて紹介したい。正しくかつ効率的にデータを整理するためには、タグデータに関する基礎知識が必要であり、また文字列加工のための道具を使いこなせるスキルも持っていなければならない。せっかくコンピュータでコンテンツを扱うのである。最大限にコン

¹(有)自由電子出版 (〒151-0053 東京都渋谷区代々木 2-22-8 イースト(株)内) Tel. 03(5358)3941

E-mail: info@jiyudensi.co.jp

¹Jiyu-denshi Publishing Co., Ltd. (22-8, Yoyogi 2-chome, Shibuya-ku, Tokyo, 151-0053)

コンピュータの能力を活用してデータを加工したい。

4.1.1. 文字列加工のための基礎知識：

タグ付けルールの知識

まずタグ付けのルールに関する知識を持とう。文字列をeコンテンツ用データに加工するということは「タグ付け」を行うということである。「2. eコンテンツに適したデータ」ですでにタグデータについて簡単に解説したが、タグデータに対する感覚をしっかり身につけることが大切だ。そのためには代表的なタグデータであるHTMLやXMLなどを題材にして、解説書を読んで理解し、また実際に自分で書いてみることなどが役に立つだろう。

ここではタグ・データで重要な入れ子構造という原則、メタ文字とエスケープという考え方について補足をおきたい。

● 入れ子構造

XMLなどの括り型のタグではタグは必ず閉じられていなければならない。同時に「入れ子」の形でなくてはならない。入れ子とはある括りと他の括りが互いにクロスしていない状態、つまり完全に括られている（括っている）状態を指す。

たとえば

```
<段落>...<強調>...</強調>...</段落>
```

が入れ子の状態であり、

```
<段落>...<強調>...</段落>...</強調>
```

は入れ子が崩れている状態である。後者は個々のタグは閉じているが入れ子が崩れているため解釈が不明瞭であり機械的処理にも向いていない。タグデータとしては悪質なものである。

● メタ文字とエスケープ文字

メタ文字とはタグ付けのために用いられる文字で一般の文字とは区別して扱わなくてはならない文字のことである。HTMLでは‘&’ ‘<’ ‘>’ ‘”’ という文字がメタ文字となる。HTMLではタグは‘<英字列>’と記述される。つまり‘<’ ‘>’で括られた英文字列はタグでありデータの本体ではない。それではデータの本体で“<a>”という文字列そのものを記述したい場合はどうしたら良いか。そのままでは“<a>”はタグと解釈されてしまう。この不都合を解消するためにHTMLでは‘<’を<、‘>’を>と書くことにしている。“<a>”は“<a>”となってタグと間違われることはない。

ところでこのルールだけでは“<”という文字列そのものは書くことができなくなる。“<”と書けば‘<’という文字に置き換わってしまうからだ。そこでもう1つルールを設け、‘&’自体を書きたいときは&と書くことにする。すると“<”

という文字列は“<”となり無事“<”を記述できる。このようにメタ文字自身を記述するためのルールをエスケープという。

タグ付けを自分で行うときにはいろいろなルール決めを行う。たとえば簡易タグ付けを行う場合で行の先頭に‘◆’があればその行は章の見出しなどと決める。しかしそのルールだけでは普通の文章の先頭に‘◆’が使えない。そんなときは事前に通常の‘◆’を他の文字でエスケープしてやれば良い。たとえば‘#’はエスケープ文字というルールを決める。こうしておけば行頭に‘◆’が来たら‘#◆’と書けば良いことになる。‘#’自身はどう書くか。‘##’と書くまでのことである^{注1}。

文字と文字コードについての知識

eコンテンツのデータ整備で最も重要で最も知識が必要とされるのが文字と文字コードの知識である。現在は世界各国個別の文字コードから世界共通のユニコードへの転換期にあたっている。ユニコードが完全に普及してしまえば問題はなくなると思われるが現状ではユニコードの実装の有無、各国コードとユニコードとのコンバートの関係など少々混乱した様相を呈している。

特に日本語のコンテンツの場合、膨大な漢字の存在、固有名詞を中心に複雑な異体字関係への対応など頭の痛い問題が多くある。文字と文字コードについては別途章を設けて解説することにする。

4.1.2. 文字列加工のための道具：

コンピュータで文字列を扱うにはそれなりの道具とそれを使いこなすスキルが必要である。これから紹介する道具はたいへん強力な機能を持っている。同時にそれらの機能ひとつひとつはとてもシンプルである。OAアプリケーションのように総合的な出来合いの機能を提供するのではなく、シンプルかつ強力な機能を組み合わせることで最高の機能を発揮する。シンプルであるから入りやす

注1) これらの‘#’は最終的には空文字と置換され、小さくなくてはならない。この場合単純に‘#’を空文字と置き換えると‘##’という表記も削除されることになる。正しくは行の頭から見ていき、‘#X’という文字列に出会ったら‘X’に置換し‘X’の次の文字に進む、というアルゴリズムで処理する。この処理は少々面倒なのでこれを簡易に行うこともできる。まず該当する文書中に存在しない文字や文字列（たとえば@を20個など）に##を仮に置き換え、その後で#を空文字に置き換え、それが済んだから最後に仮に置き換えた文字や文字列を#に置換してやれば良い。

く、しかし奥が深い。

テキスト・エディタ

● テキスト・エディタとは

文字列加工をする人の必携の道具である。テキスト・エディタはそもそもプログラムを書くための道具として発展してきた。プログラムも文章も文字列を書いたり修正したりする点は同じなのだが、一般的には文字を扱うソフトはワープロであると受け止められ、テキスト・エディタはプログラマーの使う特殊なものとして見られてきた。それではワープロとテキスト・エディタはどう異なるのであろうか。eコンテンツの制作にはなぜテキスト・エディタが良いのだろうか。

文字を書いたり修正したりする点はテキスト・エディタとワープロは基本的に差がない。しかしワープロでは画像や音声を直接埋め込んだり、何字何行という形で組版を直接実現したりできる。テキスト・エディタは文字以外は直接扱えないし、任意の組版を直接実現する手段はない。ただ文字が並ぶだけである。

この点では機能が少ないといえるが、eコンテンツのためのタグデータでは画像や音声、組版情報はタグという文字列で示さなくてはならないのである。直接画像や音声を文書中に埋め込んでも意味がないだけでなくデータとしては大問題を起こす。テキスト・エディタは純粋なテキストデータ以外を扱えない。このこと自体が、まず大切なのである（テキストデータの優位性については第1回に書いた）。

ワープロは最終的な出力を直接実現するソフト。テキスト・エディタは最終的な出力のためのデータを作ることに特化したソフトと理解すれば良いと思う。

● テキスト・エディタの特色

テキスト・エディタはテキスト以外を扱えないのだが、逆にテキストデータを扱わせたらかなうものはない。

[速い]

まずソフトを立ち上げるのが瞬時である。スクロールも速いし、置換などの速さもワープロとは比較にならない。

[大きなデータを扱える]

数10 MB程度なら問題なく扱える。ワープロでは考えられないことである。仕様のことはデータのサイズは無制限というテキスト・エディタも多い。

[文書の加工機能が豊富かつ強力]

文書を整形したり画像を埋め込んだりする機能

の代わりに、文書を加工する機能が豊富にそろっており、スピードも比較にならない。

[使い勝手が良い]

そもそもプログラマーが自分たちが使う道具として開発してきた経緯がある。使い勝手が工夫されてきたのは当然である。表示のカスタマイズ、メニューやキー割り当てのカスタマイズなど使い勝手を良くする機能が豊富にそろっている。

[強力なundo機能]

間違ってデータの一部を削除してしまったりした場合、元の状態に戻す機能。どこまで戻れるかはテキスト・エディタによって異なるがワープロなどよりはるかに前までの復元が可能である。

● テキスト・エディタの選び方

Windowsについてくる「メモ帳」も簡単なテキスト・エディタだが、機能が貧弱すぎ、ここでは対象外である。テキスト・エディタは毎日使う道具であり作業効率を大きく左右する。高機能とされるテキスト・エディタの中から選ぶことが大切である。高機能といってもフリーウェアから数千円のシェアウェア、製品版でも3万円弱で手に入る。

最近のテキスト・エディタのはやりは縦書きが可能であるとか、ルビの整形出力ができるなど本来のテキスト・エディタよりワープロに近い機能売り物にしている。これらの機能も結構だが我々の目的のためには文字列加工の機能を中心に選ぶことが大切だろう。文字列加工のためのポイントは「正規表現」が使えること、「マクロ」といわれる簡易なプログラム機能が強力なことなどであるが、高機能とされるテキスト・エディタならこれらの点はまず問題はないだろう。むしろ周囲に同じエディタの使い手がいるなど、スキルアップのための環境的要素を優先することも大切である。

どちらにしてもテキスト・エディタはシンプルだが奥の深い道具である。本稿ではテキスト・エディタの具体例にWindowsの代表的なテキスト・エディタである「秀丸エディタ」を使っている。私にとっては秀丸エディタはWindowsの黎明期からほぼ毎日朝から晩まで使い続けている道具であるが、いまだに新しい機能を発見したりする。いろいろ浮気をするより、1つのエディタをとことん使いこなすことが大切である。

Perl

PerlはLarry Wall氏が開発した文字列加工のためのスクリプト言語であり、UNIX育ちのツールだがWindowsにも移植されている。スクリプト言語と

は一種のプログラム言語であるが、大規模なアプリケーションを開発するためのプログラム言語ではなく、その場の必要に簡単に対応できることに主眼が置かれている言語を指す。

こう書くと簡易な機能しか持たないと思われがちだがPerlはたいへん強力な言語でもある。特に文字列加工にたいへん向いており、こと文字列加工に絞るなら一般のプログラム言語の敵ではないだろう^{注2}。

プログラム言語というと自分にはプログラムなどできないと敬遠する方が多い。たしかに複雑で大規模なアプリケーション開発はプロの仕事だが、その場の必要を満たすだけのプログラムは簡単なものであり我々素人でも十分に活用が可能である。

テキスト・エディタでも文字列加工は可能であるし効率も上げることができる。しかし少し込み入った処理や、フォーマットからフォーマットへの変換など繰り返し行う定型的な処理ではPerlを使えるかどうかで大きな差が出る。ぜひPerlにチャレンジしていただきたい。

残念ながら本稿ではPerlの使い方を解説する余裕はないが、WebのCGIでPerlをすでに使用されている方やこれからPerlに取り組む方のために、Perlに向いた処理では簡単なスクリプトを紹介していく。またこれから取り組む方のためにPerlの入手、インストール方法と超簡単なPerlの解説を囲み記事で書いてみた。参考にいただきたい。

● PerlとSed

文字列加工のためのスクリプト言語としてはSedが有名であり元祖といえる。Sedは置換処理を中心とした行単位の文字列処理言語であり、置換機能だけを利用するなら処理スピードも速く現在でも十分使える言語である。また機能が簡単なだけに入門用として向いていると言う人もいる。

しかしSedは機能が簡単なだけに少し複雑な処理を行おうとすると、ひどく難解なスクリプトを書くはめに陥る。こういった場合Perlを使うとSedに比べてはるかにシンプルに処理を記述できるのである。またPerlでSedの機能と同じことをさせる場合でもSedとほぼ同じような簡単さで記述できる。文字列処理言語の入門用にSedを紹介する向きもあるが、SedはPerlを実現できないがPerlはSedを簡単に実現できる。どうせ取り組むなら最初からPerlにしたほうが良い。

4.1.3. 文字列加工のためのテキスト・エディタの機能：

文字列処理のための道具としてテキスト・エディ

タを紹介したが、テキスト・エディタの機能としてぜひ身につけたい機能を紹介する。

マクロ

テキスト・エディタの機能のうち最も強力でぜひ使いこなしてもらいたい機能がマクロ機能である。マクロ機能にはキーボード・マクロと呼ばれるキー操作を記録して再現する機能と一般のプログラムと同じように一連の機能をファイルに記述して実行する通常のマクロがある。

● キーボード・マクロ

キー操作を記録して再現する機能であり簡単に習得できる。編集という作業は同じ操作の繰り返しが多いものだ。たとえば図1のような作業をよくやっていないだろうか。こういったちょっとした場面で活用できるのがキーボード・マクロである。最初はあまりスムーズにキーボード・マクロを活用できないかもしれないが、何回かトライしてコツさえつかめば、ちょっとした場面ですぐに適切な記録と再生ができるようになる。

キーボード・マクロは複雑な処理には向いていないが単純で小規模な反復作業を気軽にこなすには最適のツールである。

● マクロ

上記のキーボード・マクロに慣れてきたらその操作を保存してみよう。たとえば秀丸エディタなら[マクロ]→[キー操作の保存]で現在のキーボード・マクロがテキスト・ファイルに保存される。保存されたファイルはエディタで開いてみることができる。たとえば「カーソルを下に2回動かしてそこで“あいう”と入力した後、改行を入れる」というキー操作を記録すると以下のようなファイルができ上がっている。

```
down;
down;
insert"あいう";
insertreturn;
```

down;はカーソルの下移動、insert"～";は‘～’という文字列の挿入、insertreturn;は改行の挿入を示すマクロのコマンドである。キー操作が文字列の

注2) Perlは現在WebのCGIを記述する目的で多く使われている。そのためにPerlをCGIのためのスクリプト言語と勘違いしやすいが、Perlの本来の姿は文字列加工のための言語である。CGIへの活用は後から生まれたPerl活用の一部にすぎない。

形で記録されているから、何日経ってもこのファイルを読み込ませれば同じ作業が再現できる（秀丸エディタなら[マクロ]→[キー操作の読み込み]でキーボード・マクロに保存した操作を読み込ませることができる）。

後で再現できるだけではない。上のファイルの

“あいう”を“かきく”に変更すれば今度は“かきく”を挿入するように変更可能である。downを10回書けばカーソルが10回下に動く。これがマクロ・プログラミングの入門だ。キーボードの繰り返し操作で実現できることは検索だろうが置換だろうがマクロで書くことが可能だ。キーボード・マク

◇データ

下のファイルは簡単な住所録を入力したファイルです。1行目には名前を、2行目には住所を、3行目には電話番号が入力されています。住所や電話番号が不明の場合でも必ず空行を入れて3行に整えて入力されています。

小泉太郎
東京都千代田区永田町 X-X-X
03-1111-2222
竹中次郎
東京都中央区日本橋 Y-Y-Y

石原三郎
東京都新宿区西新宿 Z-Z-Z
03-3333-4444
...

◇加工の目的

データとしては充分ですが、人間が読むにはレコードの切れ目が分りずらく、データ処理上も簡単な手がかりが欲しいので、各行を以下のように整形したいと思います。

■名前（名前の先頭に「■」を挿入）

住所：住所（“住所：”を挿入）

電話：電話（“電話：”を挿入）

（各レコードの間に空行を挿入）

■名前

住所：住所

電話：電話

（空行）

...

◇一度手作業をやってみる

これを実際にエディタの上でやって見た上でマクロにしてみよう。

まず、ファイルの先頭にカーソルを持っていく
そこは最初の名前の先頭になっているので“■”を挿入する
次の行の先頭にカーソルを移動する
キーボードから“住所：”を挿入する
また次の行の先頭にカーソルを移動する
キーボードから“電話：”を挿入する
再度次の行の先頭にカーソルを移動する
そこは次のレコードの先頭になっているはず
そこで改行を挿入する
カーソルは依然次のレコードの先頭にあることを確認

人間ならこんな手順で作業するはず。後はこの作業の繰り返しです。でもこれを何回も繰り返すのはとても嫌な作業ですね。こういった繰り返し作業はマクロの出番です。

◇キーボード・マクロを記録する

Shift+F1キーを押してキーボード・マクロの記録を開始
以上の作業を間違えないように慎重に行う

最後まで行ったらShift+F1キーを再度押してキーボード・マクロの記録を完了

◇キーボードマクロを実行

カーソルを新しいレコードの先頭に持っていく

Shift+F2キーを押してキーボード・マクロを実行する

実行が1回終わると次のレコードの先頭にカーソルが来る

データの終わりまで再度Shift+F2キーを押してキーボード・マクロを繰り返す

うまく行きましたか？キーボード・マクロのコツはマクロの開始位置と終了位置を相対的に一致させることです。この例では「レコードの最初の行の先頭」というのが肝心の点です。こうしておけばShift+F2キーを押し続けるだけで次々に作業が進みます。

◇キーボード・マクロを書き出してみる

この程度の作業ならキーボード・マクロを繰り返すだけで十分ですが、勉強のためにマクロを書き出してみましょう。

[マクロ]→[キー操作の保存]を選択

保存のダイアログボックスでは秀丸エディタの置かれているフォルダを開ける

（デフォルト値では C:\program files\hidemaru）

任意の名前で保存する（拡張子はマクロファイルを表す .macにしておきます）

保存したファイルを開く

以上のような操作を行った場合、以下のようなファイルが記録されています。

```
insert "■";
down;
golinetop;
insert "住所：";
left 3;
down;
insert "電話：";
left 3;
down;
insertreturn;
```

insert "■";は「■」の挿入ですね。同じくinsert "住所：";は“住所：”という文字の挿入。insertreturn;は改行の挿入です。この例ではカーソルを次の行の先頭に持っていく場面で異なった表現が使われています。

ひとつは

```
left 3;
```

```
down;
```

もうひとつは

```
down;
```

```
golinetop;
```

です。

図1 3行ごとに区切られたデータを見やすくする

最初の表現はカーソルを3字左に移動して（left 3:）次に下に移動する（down:）となります。3文字を挿入した後だからこの動作で次の行の先頭にカーソルが移動します。

次の表現はまず下に移動して（down:）その後golinetop:と書かれています。実はここではカーソルを矢印キーで戻すのではなくHomeキーを使って一気に先頭に持っていきました。それがgolinetop:（行頭に移動）です。

◇記録されたマクロを検討する

どうもカーソルの移動が2つあるのが気になります。カーソルを先頭に戻す方法としてはgolinetop:の方が良さそうです。これならカーソルが何字目にあろうともカーソルを先頭に戻せそうです。書き換えると

```
insert "■";
down;
golinetop;
insert "住所: ";
down;
golinetop;
insert "電話: ";
down;
golinetop;
insertreturn;
```

となります。これがうまく動くかどうか確認していると、長い住所で行が折り返している場合にはカーソルは次の行の先頭、つまり折り返した行の先頭に行ってしまう、次のレコードの先頭には行かないことが分かりました。どうやら行が折り返している場合にはこの動作では間違いが生じるようです。

次の行（論理行）の先頭に移動するという動作をマクロを書く場合に今までの記述方法は実はすべて間違いです。正確に動作させるには次のようにします。これならどんな長い行でも間違いません。

```
golineend2;
right;
```

golineend2:というのは論理行末に移動するコマンドです。ちなみにgolineend:はキーボードのEndキーと同じで行の最後に移動します。論理行の先頭に移動するのはgolinetop2:です。

golineend2:はどんな長い行であってもその論理行の最後にカーソルを移動させます。次にright:、つまり右矢印キーを1回押せば必ず次の論理行の先頭です。

以上の修正をほどこしてマクロは以下のようにになりました。

```
insert "■";
golineend2;
right;
insert "住所: ";
golineend2;
right;
insert "電話: ";
golineend2;
right;
insertreturn;
```

レコードごとに何度もマクロを実行するのは手間ですから、ファイル1本分自動的に実行するようにします。次のマクロで一応完成です。

```
gofiletop: // ファイルの先頭に移動
while(1){ // { }の間の処理を繰り返す
    insert "■";
    goineend2; // 論理行末に移動
    right; // 右に移動
    if(result== 0)break; // 移動できなかったら（ファイル
                        // の最後）ループを抜ける

    insert "住所: ";
    goineend2;
    right;
    insert "電話: ";
    goineend2;
    right;
    insertreturn;
}
backspace; // 最後に挿入された余計な■を削除
endmacro; // マクロを終了する
```

上のコード中、while(1)は終了条件なしの無限に繰り返すループです。この無限ループを強制的に終了させるのがbreak:文です。if(result== 0)break:は直前のright:コマンドが実行できたかどうかの結果（予約変数result）を判断し、移動できなかったとき処理の繰り返しを終了（break:）しています。resultコード0は移動できなかったことを示します。右に移動できないというのはどういう意味でしょうか。つまりカーソルがファイルの最後に来ているということです。

この時点で繰り返しを終了しますが、すでに前の行で余計な「■」が挿入されてしまっています。これを取り除くのがbackspace:（Back space キーと同等）です。

なお「論理行頭に移動」とか「論理行末に移動」などの機能を持ったキーはデフォルトでは設定されていません。マクロを使いたずとたいへん良く使う機能なのでキー割り当ての機能を使ってどこかのキーに割り当てると良いでしょう。割り当てていない場合でもこの機能は使えます。[その他]→[コマンド一覧]→[カーソル移動系]とメニューを辿っていくとこの機能が選択できます。余談ですが秀丸エディタの全機能がこのメニューから実行できます。どんな機能があるか調べてみると良いですよ。

図1 3行ごとに区切られたデータを見やすくする（つづき）

ロを行ったら必ずファイルに書き出してみよう。ヘルプファイルで機能を学習することも大切だが、キーボード、マクロを書き出してみると実感を持ってマクロが理解できるはずだ。

実はマクロはもっといろいろなことができる。この段階に進んだら変数の扱いや制御文の扱いを

覚えよう。たとえば項目のIDを000010から10ずつ増加させて記入したい場合などは変数^{注3}を使えば簡単だし、もっと複雑な判断を含んだ動作をさせ

注3) プログラムの中で数値や文字列の値を覚えておく入れ物

なければifやwhileといった制御文（マクロの動作を場合によって分岐させたりある条件の範囲で繰り返したりする文）を使う必要がある。

マクロについては他人のマクロファイルの記述内容を検討してみるのが一番の勉強だ。各エディタのWebサイトには必ずマクロを集めたサイトが紹介されているはずである。そこで気に入ったマクロをダウンロードしてみよう。人のマクロを使うだけではなく、どうやってその機能を実現しているかを調べれば独力でマクロ開発もできるようになる。

Grepとタグジャンプ

● Grep

Grepはグローバル検索という検索機能を実現する。グローバル検索とは複数のファイルの中を1回で検索する機能である。1つのコンテンツが1本のファイルになっている場合には問題がないが、複数に分かれていることも多いはずだ。またコンテンツ制作以外でもこのフォルダの中のファイルからある文字列の書かれているファイルを探したいという場面にはよく出会う。こんなときは、いちいちファイルを開くのではなくGrepを使って一発で検索してしまう。Grepはそもそも単独の検索ソフトであったが、テキスト・エディタの一部の機能として組み込まれている場合が多い。

秀丸エディタを例にGrepの手順を説明する。

[検索]→[Grepの実行]でGrepのダイアログボックスが開く（図2）。

「検索する文字列」には通常の見索と同じように検索したい文字を入力する。検索文字列には正規表現も使える。その場合には下の「正規表現…」のチェックボックスをチェックする。これも通常の見索と同様である。

通常の見索と異なるのは「検索するファイル」という項目である。ここに検索対象となるファイルをワイルドカードを使って記述する。たとえばすべてのテキスト・ファイルを対象に見索するなら「*.txt」といった指定をしてやれば現在のフォルダの拡張子が“.txt”のファイルを全部見索してくれる。

このときに「サブフォルダも見索」のチェックボックスをチェックしてやれば、現在のフォルダ以下のすべての子フォルダのファイルも見索対象に含むことができる。

指定を終えて[OK]をクリックすると見索が開始される。通常の見索と異なるのは見索位置にカーソルが直接飛んでいくのではなくタグジャンプ・テーブルと呼ばれるリスト形式で見索結果が示されることである。

● タグジャンプ

図3にタグジャンプ・テーブルの例を示した。タグジャンプ・テーブルは最初にファイル名が相対

図2 秀丸エディタのgrepのダイアログボックス

位置指定で書かれる。その後の括弧内の数字はそのファイルの中の行数である。続いて「:」を挟んでその行の実際の内容が記述される。

このファイルの任意の行にカーソルを合わせタグジャンプのキー（デフォルトではF10）を押す。すると自動的に該当ファイルが開き、該当の行数までスクロールしてくれる。これがタグジャンプである。

Grepという機能も強力だがこのタグジャンプ機能もたいへん便利に使える。Grepが作成するこのフォーマットと同じフォーマットで記述すればタグジャンプ機能は使用可能なのである。たとえばJIS外字の存在する行をPerlを使って調査する場合、該当の行をタグジャンプ・テーブルの形で書き出せば修正作業は非常に楽になる。

● Grepの活用例

Grepを使うと長大な文章からある特定の行だけを取り出し一覧を得ることができる。たとえば見出しの行の行頭が必ず「■●◆◇」という記号で簡易タグ付けされていたとする。見出しの一覧は

以下のような正規表現を使えばGrepで簡単に作成できる。

^[■●◆◇]

この正規表現はすでに何度か出てきた。^は行頭を表す。[]で括られた部分はその中のどれか1文字を表す。得られたリストは図4のようなものである。このリストから「:」までの文字列を削除し（検索文字列は“^[^:]+:”，置換文字列は空文字），その後で

‘●’ は空白2つと ‘●’

‘◆’ は空白4つと ‘◆’

‘◇’ は空白6つと ‘◇’

に置換してやれば図5のようなインデントされた見出しリストが得られる。

4.1.4. 正規表現と置換：

すでに何度か正規表現と置換については書いてきた。今回は正規表現のルールについて囲み記事で簡単にまとめてみた。文字列加工では正規表現と置換が処理の中心を占めるといっても過言ではない。

```
001¥001.txt (17): 第3章 外字処理と文字コード
001¥001.txt (18): 人名・地名など固有名詞ではJIS外字の処理は大きな問題である。ユニ
コードなど新しいコード体系も視野に入れながらJIS外字についてのチェック方法や処理方法
について解説する。
001¥001.txt (98): テキスト変換で注意する点は<1>改行整形テキストにしない、<2>特殊な
フォントによる外字、<3>制御コードの出力である。
001¥001.txt (110): <2>特殊なフォントによる外字
003¥003.txt (8): PDF、CTS、電算写植、テキスト・データ、Acrobat、組版データ、データ
の整合性、組み合わせ文字、割注、ハイフオネーション、小組、外字、文字化け、トリガー・
ファンクション、OCR
003¥003.txt (57): 空白の混入は外字などが空白に置き換わったり、何かのファンクション
が空白に置き換わることが多いようである。
003¥003.txt (82): <10>外字
003¥003.txt (83): PC上のDTPシステムは基本的にJISベースで動いており、JIS外字はDTPシ
ステム上も外字である。そのため外字の発見も比較的簡単である。しかしCTSや電算写植は文
字体系がJISベースではなく独自コード体系である場合が多い。したがってテキスト化にあた
っては、JISへのコード変換テーブルが使用される3)4)。
003¥003.txt (86): <b>JIS内の文字が外字扱いに
003¥003.txt (87): どう見てもJISにある文字なのに、外字として出力されることもある。
変換テーブルの問題である場合もあるが、注意してみると細かい字体差によるものが多い。
印刷目的のシステムではたいへん多くの種類の異体字を扱えるようになっている。これが逆
に問題となる。
004¥004.txt (81): grepという機能も強力だがこのタグジャンプ機能もたいへん便利に使え
る。grepが作成するこのフォーマットと同じフォーマットで記述すればタグジャンプ機能は
使用可能なのである。たとえばJIS外字の存在する行をPerlを使って調査する場合、該当の行
をタグジャンプ・テーブルの形で書き出せば修正作業は非常に楽になる。
005¥005.txt (3): なおチェックのうち最も重要な文字に関するチェック、すなわち文字化
けやJISコード範囲、Windows外字の扱いなどは第3章でまとめて解説する。
...
```

図3 タグジャンプ・テーブルの例


```

004. txt (1): ■第2章 取り出されたテキストデータの注意点とクリーンアップ
004. txt (4): ●文字列加工のための基礎知識と道具
004. txt (6): ◆文字列加工のための基礎知識
004. txt (7): ◇タグ付けルールの知識
004. txt (21): ◇文字と文字コードについての知識
004. txt (24): ◆文字列加工のための道具
004. txt (26): ◇テキスト・エディタ
004. txt (43): ◇Perl
004. txt (52): ◆文字列加工のためのテキスト・エディタの機能
004. txt (54): ◇マクロ
004. txt (70): ◇grepとTagジャンプ
004. txt (90): ◆正規表現と置換

```

図4 見出しリストを取り出す1

```

■第2章 取り出されたテキストデータの注意点とクリーンアップ
  ●文字列加工のための基礎知識と道具
    ◆文字列加工のための基礎知識
      ◇タグ付けルールの知識
      ◇文字と文字コードについての知識
    ◆文字列加工のための道具
      ◇テキスト・エディタ
      ◇Perl
    ◆文字列加工のためのテキスト・エディタの機能
      ◇マクロ
      ◇grepとTagジャンプ
    ◆正規表現と置換

```

図5 見出しリストを取り出す2

正規表現はルール自体はさほど複雑ではない。しかし必要なパターンを正規表現で素早く正確に書けるようになるには経験の積み重ねが必要である。スキル向上には他の人の書いた正規表現を見ることが大切である。あまり数は多くないが正規表現を取り上げた本も出版されているので読んでみることも良いだろう。

置換は強力な機能である。確認付きの置換なら間違いは少ないだろうがPerlなどを利用した置換や確認なしの置換では間違えて置換を行いデータの一部を消失してしまうことも経験するだろう。Perlなどでは元ファイルが残るからやり直しもきくが、

テキスト・エディタで直接置換した場合には元に戻すことはたいへん困難だ。いくら強力なundo機能があっても置換で一気に処理したものを順に戻すとなるとたいへんな作業が発生する^{注4)}。ごく基本的なことだが一定のステップごとにバックアップ・ファイルを残すようにすることが大切である。

注4) 秀丸エディタには編集メニュー中に「最初からやり直し」という機能もある。これは最後に保存した内容にすべて戻してくれるのだが、逆にいえば更新した内容はすべて消えてしまうという取り扱い注意のコマンドでもある。

超簡単なPerlの解説

ここではこれからの連載記事を理解するために必要なPerlの知識を説明します。イメージをつかんでいただくためのものですから、細かい規則などには触れていません。Perlを解説した本はいろいろ出版されていますが、UNIXをベースに解説したものが多く、Windowsユーザは選択に注意してください。Windowsでの入門用としては『初めてのPerl Win32システム』（オライリー・ジャパン）などが良いと思います。

● 日本語対応Perl

PerlはGNUのフリーウェアとして流通しています。Perlも5.6以降ではユニコード対応（UTF-8）が進んでいるようですが現状では十分安定していません。文字列加工のためには日本語対応Perl（JPerl）がお勧めです。

Windows環境でJPerlをインストールするためには、最初に英語版のPerlをインストールし、その後で日本語化したJPerlをインストールするという2段階を踏みます。

Perlの入手法やインストール手順は「Perl初心者の部屋」<<http://www.harukaze.net/~mishima/perl/index.html>>の中の「Windows2000/NT/95/98で日本語Perl（JPerl）」<<http://www.harukaze.net/~mishima/perl/win/jperl.html>>に詳しく紹介されています。「Perl初心者の部屋」はたいへん充実したページです。BBSの全文検索などがありますのでPerlプログラミングでつまづいた場合に大いに参考になるでしょう。

ソフトウェアのダウンロードサイトVector<<http://www.vector.co.jp/>>にも用意されています（場所はWindows>ユーティリティ>テキストファイル用>Perl）。Vectorのテキストファイル用ユーティリティのページには文書加工用ツールがいろいろあるのでのぞいてみると良いでしょう。

● Perlの使い方

Perlはコマンドプロンプトから起動します。処理内容はPerlスクリプトと呼ばれるテキストファイルに書きます。abc.plというPerlスクリプトを実行するには以下のようなコマンドを使います（日本語Perlはjperlというコマンドです。最初の「>」はプロンプトですので入力不要です）。

```
>jperl abc.pl
```

Perlのスクリプト・ファイルの拡張子は通常は“.pl”とします。Perlは実行するとまずスクリプトをコンパイルし、その後で処理を開始します。試しにテキスト・エディタで次の1行だけのファイルを作成し、好きな名前、たとえばfirst.plという名前で保存してください。これでスクリプトファイルは完成です。

```
print "Hello Perl!!\n";
```

次にコマンドプロンプトをスクリプトファイルのあるフォルダに移動し、

```
>jperl first.pl
```

と入力して実行すると“Hello Perl!!”と画面に出力されるはずです。

● 変数

Perlの変数は「\$」に続けて英数を書くことで表します。英数は大文字小文字を区別します。Perlでは数値型と文字列型の区別はありません。スクリプトの文脈で自動的に変換されます。

上のごあいさつスクリプトを変数を使って書くと以下になります。

```
$shello="Hello Perl!!!";
```

```
print $shello."\n";
```

‘.’は文字列を結合させる演算子です。改行文字“\n”を後ろに付けています。

● ファイルの扱い

Perlはファイルハンドルを使って複数のファイルを、いろいろなモードで開くことができますが、標準入力、標準出力を使用するのが簡単な方法です。次の例はABC.txtをabc.plで読み込み、ABC2.txtに出力するコマンドラインの入力例です。

```
>jperl abc.pl ABC.txt >ABC2.txt
```

● 行指向プログラミング

文字列処理の場合スクリプトの基本はテキスト・ファイルを1行ずつ読み込み、任意の処理を施したうえで出力するものです。これを行指向のプログラミングなど呼びます。Sed、Awkといったツールと同じですね。下の例は単なるコピーです。

```
while(<){
    print;
}
```

while()内の条件が成立している間{ }内の処理を繰り返し行います。()内に書かれている“<”は標準入力から1行ずつ読み込むことを表します。最後の行まで読み込むと“<”が偽となり、このループは終了します。“print”は読み込んだ文字列を標準出力に出力します。

このwhile(<){ }の間にさまざまな処理を書くことになります。

chomp;

行末の改行文字を取り去ります。行末の改行文字があると、文末でのマッチや行の連結などに不便なことが多いので、基本的にchompコマンドで改行文字を取り去ってから処理をすると良いでしょう。

この場合、出力させる場合には改行文字を付加することになります。以下のスクリプトは上のコピースクリプトと同じ動作となります。

```
while(<){
    chomp;
    print "$_\\n";
}
```

新たに“\$_”という変数が登場しました。“<”で読み込んだ内容は“\$_”という特殊変数に自動的に格納されます。各コマンドの変数パラメタが省略されたときはこの変数を対象に処理が行われます。chompもパラメタが省略されていますから、\$_から行末の改行文字を削除しています。出力では改行文字を付加するために“\\n”を付けています。

Perlではダブルコーテーションで括られた文字列中に直接変数を記述できます“\$_\\n”。記述した変数、この場合は“\$_”はPerlが展開して出力してくれます。シングルコーテーションで括るとこの展開は行われません。

● 置換

置換は以下のようなs演算子を使って行います。

s/被置換文字列/置換文字列/;

被置換文字列は正規表現として解釈されます。以下は“です。”を“だ。”に“します。”を“する。”に置換をします。

```
while(<){
    chomp;
    s/です。/だ。/;
    s/します。/する。/;
    print "$_\\n";
}
```

置換のオプション

s演算子の最後に‘g’を付けると1行に複数のマッチ箇所があった場合にすべてについて置換します。付けない場合は最初にマッチしたところだけが置換されます。上に書いた置換のスクリプト例、本当は以下のように‘g’を付けないと一部しか置換してくれません。

s/です。/だ。/g;

s/します。/する。/g;

また‘i’を付けると、英字の大文字、小文字を区別しないでマッチします。次の例はABC, ABc, Abc, abc, aBC…にマッチして“エイビーシー”に置換します。

s/ABC/エイビーシー/i;

● 変換

trは大文字と小文字の変換、カタカナとひらがなの変換など一対一の置き換えを行います。置き換えは‘/’の前後の文字列の対応している文字に置き換えられます。次の例は文字列中の文字‘a’を‘A’に、‘あ’を‘ア’に、‘c’を‘C’に置き換えます。

tr/aあc/AアC/;

置き換える文字列がコード上連続して対応する場合は‘.’を使って簡単に書くことができます。次の例はアルファベットの大文字化、ひらがなのカタカナ化をします。

tr/a-z/A-Z/;

tr/あ-ん/ア-ン/;

trはSedのyコマンドと同じです。

● マッチ演算子

検索文字列をマッチさせます。sのように文字列を変化させることはありませんが、マッチすれば真をマッチしなければ偽を返しますので条件判断に使われます。また次の後方参照でよく使用されます。

次の例は「〒郵便番号(空白)住所」とフォーマットされた行にマッチします。なおマッチ演算子はmを省略して単に/~/と書くことができます。

m/^[〒0-9]{3}(-[0-9]{4})?[]+\$/;

マッチ演算子に‘g’のオプションを付けると、行中でマッチした複数の部分がすべて配列の形で取り出せます。次のスクリプトは<key>というタグをすべて取り出し出力します。

```

if(@list=m/<key>.*?<¥key>/g){
    foreach $key(@list){
        print "$key¥n";
    }
}

```

if(@list=m/<key>.*?<¥key>/g)という部分はPerl独特の書き方です。次のような2行を1行で書いてしまいます。分かりやすく下のように書くこともお勧めです。

```

if(m/<key>.*?<¥key>/){
    @list=m/<key>.*?<¥key>/g;

```

“<key>.*?<¥key>”に行がマッチするとif文が成立して{ }内を実行します。同時に@listという配列にマッチした部分に取り込まれます。{ }内は@listの内容を全部出力しているだけです。

● 後方参照

基本的ルールは正規表現の囲み記事を参照してください。

マッチ演算子,s演算子でマッチさせた場合,マッチ文字列の全体は‘\$&’という変数に格納されます。次のスクリプトは先頭が‘■’で始まりもう1つ‘■’がある行を取り出し, ■から■までの間の文字を出力します。

```

if(m/^■.*?■/){
    $a=$&;      # “■～■”を変数$aに格納
    $a=s/■//g;  # ■を削除する
    print "$a¥n"; #出力する
}

```

“\$a=”という見慣れない表現が登場しました。sコマンドやmやtrコマンドは何も付けなくて使うとデフォルトの特殊変数\$_を対象に置換やマッチ、変換をしますが, “=”という演算子を使うと特定の変数に対して置換, マッチ, 変換をすることができます。ここでは\$aという変数に納められた文字列中の‘■’を空文字に置換しています。

検索文字列が()でグループ化されている場合は\$1, \$2…という変数に内容が格納されます。次の例は前述の住所録のフォーマットを変更します。

```

if(m/^( [0-9]{3}( - [0-9]{4})? ) [ ]( + )$/){
    print "郵便番号：$1¥n";
    print "住所：$3¥n";
}

```

この正規表現は()が入れ子になっています。こんな場合でも‘(’の現れる順で\$1, \$2と番号が振

られますから, 郵便番号全体は\$1, 住所部分(空白以下)は\$3になります。\$2は郵便番号のハイフン以下になるはずですが。

● 比較演算子

数値用の関係演算子は以下です。他の言語と同じですから特に問題はありませんが, “<=”や“>=”を“=<”“=>”と書くと他の演算子になってしまいます。注意してください。

==	等しい
!=	等しくない
<	小さい
>	大きい
<=	以下
>=	以上

Perlには文字列用の関係演算子が用意されています。Perlは変数の型として数値と文字の区別がないので, 文字列として評価したい場合には必ずこの専用比較演算子を使います。

たとえば‘001’と‘1’は数値として評価した場合は等しいのですが, 文字列としては‘001’のほうが小さくなります。

eq	等しい
ne	等しくない
lt	小さい
gt	大きい
le	小さいか等しい
ge	大きいか等しい

● 配列変数

配列

複数の値をセットで持ちます。“@+英数”で表現します。セットは0番からn番まで持つことができ, 番号で個々の値を呼び出します。次のスクリプトは月の英略称を配列に納めています。

```

@month=("Jan","Feb","Mar","Apr","May","June","July","Aug","Sept","Oct","Nov","Dec");

```

個々の値を使用する場合には“\$+配列名[番号]”という形で使います。次の例は“May”を出力しています。5月は5番目に格納されていますから, 配列の添え字は5-1つまり4ですね。

```

print "$month[5-1]¥n";

```

配列は()を使って定義する他に個々の値に直接代入することでも定義できます。次のスクリプトは5月の配列要素を和名にしています。

```

$month[4]='皐月';

```

ハッシュ配列

配列が数字で複数の値を管理するのに対して、ハッシュ配列は文字列をキーとして複数の値を管理できます。ハッシュ配列は“%+英数”で表現します。次のスクリプトは月の和名と英文名をセットにしています。定義の仕方は通常の配列と同じ書き方ですが、ハッシュではキーとその中身を対で記述していきます。

```
%month=(
  "Jan","睦月",
  "Feb","如月",
  "Mar","弥生",
  "Apr","卯月",
  "May","皐月",
  "June","水無月",
  "July","文月",
  "Aug","葉月",
  "Sept","長月",
  "Oct","神無月",
  "Nov","霜月",
  "Dec","師走");
```

ハッシュ配列の個々の要素はキー文字列を{ }で囲って呼び出します。たとえば、

```
print "$month{'June'}\n";
```

とすると“水無月”が出力されます。

ハッシュ配列は一種のデータベースのような使い方が可能で、さまざまな活用法があります。

ハッシュ配列は()を使って定義する他に個々の値に直接代入することでも定義できます。次のスクリプトはキー“Jan”の中身を“正月”に変更しています。

```
$month{'Jan'}='正月';
```

キーは初めて現れたときに新たに自動的に作られます。次は新しいキー“10月”を作成しています。

```
$month{'10月'}='神無月';
```

キーがセットされているかどうかを調べるにはexistsという関数を使います。

```
if(exists $month{'June'}){
    定義されていた場合の処理
}
else{
    定義されていなかった場合の処理
}
```

● 制御文

Perlの代表的な制御文は以下です。

if文 elsif文 else文

条件判断をして処理を分岐させます。次の例は“=”文字が登場した行を出力します。

```
if(/=/){
    print "$_ \n";
}
条件が合わなかった場合に行う処理はelse文以下に書きます。
if(/^$/){
    print "空行です\n";
}
else{
    print "空行ではありません\n";
}
```

複数の条件を順次適用したい場合は以下のようにelsif文を使います。次のスクリプトは見出しの先頭が“■●◆◇”という記号で分けられて記述されている文章からインデントされた見出しリストを作成します。

```
if(/^■/){          #先頭が■の場合
    print "$_ \n";
}
elsif(/^●/){       #それ以外で先頭が●の場合
    print "    $_ \n";
}
elsif(/^◆/){       #それ以外で先頭が◆の場合
    print "        $_ \n";
}
elsif(/^◇/){       #それ以外で先頭が◇の場合
    print "            $_ \n";
}
```

while文

ある条件が成立するかぎり処理を繰り返します。次のスクリプトは文字列の後ろに80字になるまで“.”という文字を結合します。

```
$a="abcde";          #任意の文字列
while(length($a)<80){ #length( )は文字列の長さを返す
    $a.=".";         #.= は左辺に右辺の内容を結合する
}
```

last文

繰り返しを強制的に終了します。次のスクリプトは“END”という文字が登場したら読み込みを中断します。

```
while(<>){
    if(/END/){last;}
    処理…;
}
```

next文

以降の処理を行わず強制的に次の繰り返しに飛ばします。次のスクリプトは空行を出力しないでコピーをします。

```
while(<>){
    if(/^$/){
        next;
    }
    print;
}
```

for

一定の繰り返しをさせる場合に使用します。forは制御用の変数を使用し、その初期値、繰り返し条件、増減条件を指定して使用します。次のスクリプトは1から100までの累計を出力します。

```
$add=0;          #累計値を入れる変数
for($i=1; $i<=100; $i++){
    $add+=$i;     # $addに $iの値を加える
    print "$i:$add\n";
}
```

forの行の意味は、制御変数は\$i、\$iの初期値は1、\$iが100になるまで処理を繰り返す、\$iは繰り返しの度に1増加する(\$i++)ということです。増分が1ですからこのループは100回繰り返されます。

foreach

配列の要素をすべて使う場合に使用されます。

```
foreach $i(@array){
    print "$array[$i]\n";
}
```

と書きますと、@array配列の要素を順に\$iに格納し、1行ずつ出力します。

ハッシュ配列の要素すべてを使用する場合は以下のようにします。

```
foreach $key(sort keys %hash){
    print "$key : $hash{$key}\n";
}
```

上のスクリプトは、%hashというハッシュ配列のキーをkeysという関数で取り出し、そのキーをsort関数で並べ替えたうえで順次\$keyという変数に格納し繰り返し処理を行います。{ }中のprint文で「キー：キーの値」が1行ずつ出力されます。

正規表現の基本的なルール

● 正規表現とは

正規表現とは文字列のパターンを表現する表記方法です。

たとえば東京の電話番号は03の市街局番と4桁の市内局番、4桁の番号で構成されます。それぞれの番号がハイフンでつながれていた場合、0～9までの数字は正規表現で“[0-9]”と書けますから東京の電話番号は以下のように書くことができます。

```
03 - [0-9][0-9][0-9][0-9] - [0-9][0-9][0-9][0-9]
```

正規表現を使えば文字のパターンで検索や置換が行えますので文字列処理には不可欠の道具です。

ここでは記事の解説を理解する目的でよく使われる正規表現の基本ルールを簡単にまとめてみました。正規表現には他にも便利な指定方法がありますが、以下の基本ルールの組み合わせでほとんど用が足りると思います（正規表現はいろいろなツールによって多少異なっています。マニュアルなどで確認しましょう）。

● 正規表現のメタ文字

以下の文字は正規表現中では特別な意味を持ちます。左にメタ文字を右に意味を書いてあります。

.	改行を除く任意の1文字
[]	文字クラス
^	行頭、文字クラスの否定
\$	行末
*	直前のパターンの0回以上の繰り返し
+	直前のパターンの1回以上の繰り返し
?	直前のパターンが0回または1回現れる
{ }	繰り返しの回数
()	パターンのグループ化
	パターンの論理和
\	エスケープ文字

● エスケープ文字

これらのメタ文字自身を検索したいとき、たとえば‘\’という文字を検索したい場合はエスケープ文字‘\’を使います。

正規表現（せいきひょうげん）

とすると、

正規表現せいきひょうげん

にヒットしてしまい、

正規表現 (せいきひょうげん)

にはマッチしません。

正規表現¥ (せいきひょうげん¥)

とエスケープする必要があります。

‘¥’ 自身を表したいときは “¥¥” と書きます。

● 任意の文字

‘.’ は改行を除く任意の1文字にマッチします。

日本語エディタやJPerlでは1バイト文字も2バイト文字も1文字として扱われます。英語版Perlでは2バイト文字は2文字に分解されるので注意が必要です。

● 特殊な文字表現

¥n 改行文字にマッチします

¥t タブ文字にマッチします

● 文字クラス

あるグループに含まれる文字1文字を表すことができます。

[ABC]

は ‘A’ または ‘B’ または ‘C’ にマッチします。

[ABCabc]

なら小文字の場合もマッチします。

‘.’ でつなげて文字コードの範囲を指定することができます。

[0-9]

は、文字コードで0x30から0x39の範囲の文字、つまり “0123456789” の各文字にマッチします。

[あ-ん]

は、ひらがなにマッチします。

文字コード順ですから意味では順番でも文字コードがばらばらである漢数字のような場合には使えません。

[〇-九] ×

[〇一二三四五六七八九] ○

文字クラスの範囲表現は組み合わせて使えます。

[0-9A-Za-z] 0～9とA～Z, a～zの中の1文字

[ァ-ヴー] カタカナの範囲, ァ～ヴとー (音引き)

文字クラス中のメタ文字

‘[’ と ‘]’ の間ではメタ文字は原則的に通常の文字と同じものと解釈されます。ただし、以下の文字自身を記述する場合には ‘¥’ をつけてエスケープする必要があります。

‘^’ ‘.’ ‘]’ ‘¥’

定義済み文字クラス

Perlではよく使われる文字クラスをメタ文字として定義しています。以下にあげるのはその一部です。

¥d 数字 [0-9]と同じ

¥w 単語構成文字 [_a-zA-Z0-9]と同じ

¥s 空白文字 [¥t¥n¥f¥r]と同じ

¥S 空白文字以外 [^¥t¥n¥f¥r]と同じ

● 文字クラスの否定

‘^’ を先頭に付けるとあるグループに含まれる文字ではない文字1文字を表します。

[^ABC] ABC以外の文字にマッチします。

● 位置指定

‘^’ は行頭を表します。以下の表現は行頭が第1章から第9章で始まる行にマッチします。

^第[1-9]章

‘\$’ は行末を表します。以下の表現は行末が「。」以外の行にマッチします。

[^。]\$

● 繰り返し指定

直前の文字パターンの繰り返しを表します。

‘+’ は直前のパターンの1回以上の繰り返しを表します。次の表現は空行以外の行にマッチします。

^.+ \$

‘*’ は直前のパターンの0回以上の繰り返しを表します。0回以上の繰り返しとは、0回つまり存在しない場合も含みます。“A*B” は “B” “AB” “AAB” “AAAB” … にマッチします。以下の例は空白の有無に関わらず第n章から始まる行にマッチします。

^第[_□]*[0-9]+[_□]*章

注: _ は半角の空白, □ は全角の空白を表している。

‘?’ は直前のパターンが0回または1回現れることを表します。次の例は語尾の音引きや中黒の有無を無視してマッチします。

コンピューター?・?システム

Perlなどでは{ }を使って繰り返しの回数を指定できます。

A{2,4} AA～AAAAにマッチします。

“AAA?A?” と同じです。

A{2,} Aが2回以上繰り返す “AA+” と同じです。

A{2} Aが2回つまりAAにマッチします。

これを使って日本の電話番号を表現すると以下のようになります。

0[0-9]{1,3} - [0-9]{2,4} - [0-9]{4}

● パターンのグループ化

()を使ってパターンをグループ化できます。次項の論理和でよく使います。

→(別項)?参照

上の表現は“→別項参照”にも“→参照”にもマッチします（このようなグループに繰り返し指定を付ける表現は秀丸エディタでは使えないようです）。

● 論理和

複数の候補のいずれかであった場合にマッチします。たとえば

天候：(曇り|曇|くもり)

上の表現は“天候：曇り”“天候：曇”“天候：くもり”にマッチします。

● 後方参照

Perlでは()で括られた部分にマッチした文字列を‘\’が登場する順番で¥1, ¥2という表現で表わし同じ正規表現中で利用できます。たとえば

<(H[1-5])>.+</¥1>

上の表現は“<H1>～</H1>”“<H2>～</H2>”…“<H5>～</H5>”，にマッチします。

また、正規表現の外側たとえば置換などでも、\$1, \$2という変数にマッチした部分を格納して後方参照が使えます。perlでは置換を“s/検索文字列/置換文字列/”と書きます。

s/(王様)は(裸)だ/\$2は\$1/;

と書けば“裸は王様だ”に置換されます。

s/<title>(.+)</title>/\$1/;

では<title>タグの中身が取り出せます。

通常の変数としても使えます。これは行の要素を取り出すのに最適です。“m/検索文字列/”は検索文字列にマッチしたときに真を返します。下のスクリプトはtitleタグのある行にマッチしてその中身だけを一覧として出力します。

```
if(m|<title>(.+)</title>|){
    print "$1¥n";
}
```

タグ付き置換

秀丸エディタでは正規表現を‘¥f’で区切ることによって、マッチした部分文字列を置換文字列で使うことができます。¥fで区切られた順に¥0, ¥1, ¥2…となります。Perlとは番号が0から始まる点が異なりますので注意してください。

検索文字列：王様¥fは¥f裸¥fだ

置換文字列：¥2は¥0だ

検索文字列：<title>¥f.+¥f</title>

置換文字列：¥1

● 最長一致，最短一致

‘*’や‘+’などの繰り返し指定は最長のパターンにマッチします。たとえば次のような文章から()部分を削除する場合を考えます(ここで使用している丸カッコは全角ものなので‘¥’でエスケープしていません)。

正規表現(せいきひょうげん)を使(つか)いこなせると文章(ぶんしょう)の加工(かこう)が楽(らく)だ。

これを検索文字列“(.)”，置換文字列空文字という置換をかけると

正規表現だ。

になってしまいます。“+”の部分、つまり任意の文字の連続が最長になるようマッチすると以上のような結果になることが分かりますか？

Perlでは‘*’や‘+’に‘?’を組み合わせると最短一致が使えます。上の例ですと、

s/(.+?)//g;

とすれば、それぞれのカッコに対応します。

この方法を使わなくとも以下のようにすれば、上記の場合の最短一致は実現できます。この正規表現は秀丸エディタでも使用可能です。閉じ丸パーレンでない文字の連続と指定してやるわけです。

s/([^\s]+)//g;