知識情報演習 III – Perl 入門 –

辻 慶太

http://slis.sakura.ne.jp/cje3/

1 とにかく実行しよう

Perlプログラムの作成から実行までの手順を示す.

1. エディタ（emacs など）でプログラムを編集する．ここではプログラム名を「a.prl」とする．Perlではプログラム名に関する決まりはないと書いた．実際には，慣習として「.pl」を付けることが多い．しかし，Prolog という別の言語ではプログラムの拡張子を「.pl」としなければならない．Prolog も使う人は両者を混同しないよう注意しよう．
2. プログラムを実行する．

```
% perl a.prl
```

例題 1: 以下のプログラムを入力，実行しよう.

```
1: while ($line = <STDIN>) {
2:     chomp($line);
3:     print "your input: $line\n";
4: }
```

例題 1 の解説

1: ある手続きを繰り返し実行する「while」ループの始まりを表す．括弧内は，キーボードからの入力（STDIN）が続く間，それを変数「$line」に代入する，という意味である.
2:「chomp」は文字列の改行文字「\n」を削除する.
3:「print」を用いて，キーボードから入力された文字を表示する.

※ 例題 1 のプログラムは「Ctrl+D」で終了する.

例題 2: 以下のプログラムを入力，実行しよう.

```
1: while ($line = <STDIN>) {
2:     chomp($line);
3:     if ($line eq 'bye') {
4:         print "さようなら\n";
5:         last;
6:     } elsif ($line eq 'hello') {
7:         print "こんにちは\n";
8:     } else {
9:         print "あんた無愛想だね\n";
10:    }
11: }
```

例題 2 の解説

3: $line が「bye」に等しいかどうか調べる.
5:「last」は，その時点でループを終了する.
6: $line が「hello」に等しいかどうか調べる.
9: 3 行目，6 行目のどちらの条件にも当てはまらない場合は，ここが実行される.

例題 3: 以下のプログラムを入力，実行しよう．キーボードからは「21 3.5 -7」のように空白で区切った 3 つの数値を入力すること.

```
1: while ($line = <STDIN>) {
2:     chomp($line);
3:     @a = split(/ /, $line);
4:     $x = $a[0] + $a[1] + $a[2];
5:     print "$x\n";
6: }
```

例題 3 の解説

3:「split」は文字列を分割する．第 1 引数「/ /」（半角スペースを入れるように）は「空白で分割せよ」という意味．分割した結果は「@a」という配列に代入される.
4: 配列「@a」の最初の 3 つの要素を足して「$x」に代入する．配列の添字は 0 からはじまる．また，配列の要素を参照するときは，先頭に「@」ではなく「$」をつける点に注意.

例題 4: 以下のプログラムを入力，実行しよう.

```
1: @a = (6, 2, 8, 3, 1, 4, 7);
```

```
2: print "ソート前: @a\n";
3: @a = sort @a;
4: print "ソート後: @a\n";
```

例題 4 の解説

1: 配列要素の代入．丸括弧を使う点に注意しよう．
2:「print "@a\n"」は，配列要素を全て出力する．
3:「sort」は配列要素をソートする機能である．

例題 5: 以下のプログラムを入力，実行しよう．これは（キーボードから）入力された名前に対応する ID を出力するプログラムである．

```
1: %id = ();
2: $id{'kuma'} = 'x1';
3: $id{'neko'} = 'x2';
4: $id{'kiji'} = 'x3';
5: $id{'saru'} = 'x4';
6: while ($line = <STDIN>) {
7:     chomp($line);
8:     print "$id{$line}\n";
9: }
```

例題 5 の解説

1: ハッシュ「%id」の初期化
2: ハッシュ要素の代入．ハッシュとは，数値以外のデータを添字（キー）にできる配列である．キーは「{}」で括る点に注意．また，文字列定数はシングルクオート「'」で括る点にも注意．
8: 入力されたキーに対応する値を出力する．

2 Perlプログラムの形

2.1 概要

- 文の最後はセミコロン「;」で終わる．
- 変数の前には「$」，配列の前には「@」，ハッシュの前には「%」をつける．
- 変数の宣言は（原則として）必要ない．
- 「#」以降は行末までコメントとして無視される．

2.2 条件分岐

テンプレートを以下に示す．ここで「else if」ではなく「elsif」である点に注意を要する．

```
if (条件式 1) {
    条件式 1 が成立したときに実行する命令;
} elsif (条件式 2) {
    条件式 2 が成立したときに実行する命令;
...
} else {
    どれにも該当しない場合に実行する命令;
}
```

2.3 繰り返しの制御

- while
  ( ) 内の条件が満たされる間，{ }内を繰り返す．( ) 内の条件がいつまでも満されなければ「無限ループ」に陥る．

  ```
  $x = 0;
  while ($x < 100) {
      print "$x\n";
      $x++;
  }
  ```

- foreach
  主に配列やハッシュの要素を順番に処理する場合に用いる．以下は，配列「@a」の要素を順番に出力する．

  ```
  foreach $i (@a) {
      print "$i\n";
  }
  ```

  以下は，2 から 5 までの数字を順番に出力する．

  ```
  foreach $i (2..5) {
      print "$i\n";
  }
  ```

  いずれの例も，括弧内の値が順番に変数$i に渡される点に注意しよう．

- last
  現在のループを脱出する．ループの種類は，while とforeach のどちらでもよい．

- next
  現在の回を中断して，次の回を実行する．ループの種類は，while とforeach のどちらでもよい．

3 データ型と使用例

3.1 定数と変数

変数をクオートで括るときは，シングルクオート「'」とダブルクオート「"」の違いに注意すること．前者は括られたもの「そのもの」を指し，後者は括られた変数の中身を指す．定数の場合，どちらで括っても実害はない．

例題 6: 以下を実行して違いを実感しよう.

```
$i = 'hoge';
print "$i\n"; # hoge と表示される
print '$i\n'; # $i\n と表示される
```

3.2 配列の使用例

以下, 配列に対する主な操作の例を載せる. ただし, 完全なプログラムではないため適宜補うこと. 特に, 変数や配列の値を print で随時表示すると, 何が起こっているのかが分かりやすい.

- 配列への代入
```
@a = (2, 6, 10);
@b = ('aaa', 'bbb', 210, 'ccc');
```
- 配列どうしの代入
```
@a = (2, 6, 10);
@b = @a;
```
- 追加と削除
```
@a = (2, 6, 10);
@b = (1, 5, 7);
$x = -3;
push(@a, $x);                末尾に追加
$y = pop(@a);                末尾を削除
unshift(@a, $x);             先頭に追加
$y = shift(@a);              先頭を削除
push(@a, @b);          配列に配列を追加
```
- 配列要素から変数への代入
```
@a = (2, 6, 10);
@b = (-5, 4, 9, 1);
($x) = @a;              先頭要素が代入される
($x, $y) = @b;    先頭から 2 つ代入される
($x, @c) = @b; 先頭以外の要素は@c へ代入
```
- 配列の要素数を調べる
```
@a = (-5, 4, 9, 1);
$x = @a;       左辺が ($x) でない点に注意
```
- 配列のソート
```
@a = (100, 22, -4, 19, 2, -31);
@x = sort @a;           アルファベット順
@x = sort {$a <=> $b} @a;      小さい順
@x = sort {$b <=> $a} @a;      大きい順
```
ここで, {$a <=> $b} や {$b <=> $a} における $a や $b は「予約」されている. そこで, これらの変数に対して値を操作することには意味がない.
- 配列要素を逆順にする
```
@a = (-5, 4, 9, 1);
@a = reverse @a;
```
- 文字列を分割した結果を配列に代入
```
$x = 'He went to school.';
@a = split(/ /, $x);          空白で分割
$x = "He\twent"; タブ（\t）入りの文字列
@a = split(/\t/, $x);         タブで分割
```
この操作は, 空白やタブで区切られた,「表」のようなファイル行を処理するときに有効である.
- 配列要素を結合して一つの文字列にする
```
@a = ('I', 'go', 'to', 'school');
$x = join(' ', @a);       空白でつなげる
$x = join('', @a);      そのままつなげる
$x = join("\t", @a);      タブでつなげる
```
タブ（\t）をダブルクオートでくくる点に注意しよう.
- 配列要素の切り出し

@x = splice(@a, 5, 3); 5 番目から 3 個分を抽出

このとき, @a の内容は変更される.
- 条件に合う要素の抽出
```
@a = (100, 22, -4, 19, 2, -31);
```
@x = grep(($_ > 10), @a); 10 より大きい要素だけ抽出
```
@x = grep(($_ <= 5), @a);
```
5 以下の要素だけ抽出

例題 7: 以上の操作を一通り試してみよう. 配列の要素は任意に設定してよい.

3.3 ハッシュ

- キーと値の代入 (例題 5 の 2 〜 5 行目と同じ意味)
```
%a = ('kuma' => 'x1', 'neko' => 'x2',
      'kiji' => 'x3', 'saru' => 'x4');
```
- ハッシュどうしの代入
```
%a = %b;
```
- ハッシュを用いた多次元配列
```
$a{2, 3} = 10;       (2 行 3 列に 10 を代入)
$a{2, 3, 1} = -5;          (3 次元の場合)
$a{'taro', 'math'} = 90;
$a{'taro', 'english'} = 70;
$a{'taro', 'jiro'} = 'brothers';
```

```
$a{'taro', 'hanako'} = 'couple';
```

なお，Perl の「レファレンス」という機能を用いて多次元配列を実現することもできる．興味がある人は自分で調べよう．

- キーによるソート

```
%x = ('kuma' => 70, 'neko' => 50,
      'kiji' => 60, 'saru' => 90);
foreach (sort keys %x) {
    print "$_\t$x{$_}\n";
}
```

- 値によるソート

```
%x = (... 上と同じ ...);
foreach (sort {$x{$b} <=> $x{$a}}
    keys %x) {
    print "$_\t$x{$_}\n";
}
```

※ 「$_」は特殊な変数であり，本来指定されるべき変数が省略された場合に自動的に使われる．

例題 8: 上記「キーによるソート」と「値によるソート」をそれぞれ試してみよう．

4 ファイル入出力

- 標準入力
  「<STDIN>」は，標準入力から 1 行読み込む．単に「% a.prl」を実行すればキーボードからの入力になり，また「% perl a.prl < file」を実行すれば「file」の内容を読み込む．プログラムでの使用例は，例題 1 を参照されたい．
- 便利な「<>」
  「<>」は，コマンドラインで指定されたファイルから 1 行読み込む．例えば，

```
% perl a.prl file1 file2
```

を実行すると，2 つのファイルの内容を順番に読み込む．ファイルはいくつ指定してもよい．標準入力の場合と違い「<」がいらない点に注意しよう．以下は使用例である．

```
while ($line = <>) {
    chomp($line);
    ...
    print "$line\n";
}
```

ファイル名が省略された場合は「<STDIN>」と同じ動作をする．

- ファイルから「イッキに」読み込む
  以下を実行すると，ファイルから全ての行をイッキに読み込み，配列@a に代入する．

```
@a = <>;
```

以下は，入力したファイル行末に付いている改行を全て削除する．

```
chomp(@a = <>);
```

- ファイルへの出力
  「print」を使って表示した内容を（画面ではなく）ファイルに出力するには，コマンドラインで以下を実行すればよい．

```
% perl a.prl > file
```

- 「open」によるファイル入力

```
$file = 'hoge';
open(IN, $file) || die "$file: $!";
while ($line = <IN>) {
    chomp($line);
    ...
}
close(IN);
```

- 「open」によるファイル出力

```
$out = 'hoge2';
open(OUT, "> $out") || die "$out: $!";
print OUT "kuma\n";
close(OUT);
```

上記のプログラムを実行する前に「hoge2」というファイルが存在する場合は，ファイルの内容が上書きされることがあるので中止しよう．

- 「open」の使用に関する注意点
  - 「IN」や「OUT」はファイルハンドルと呼ばれる．
  - 「open」を用いてファイルをオープンした場合は，使用後に「close」を用いてクローズすること．
  - 「||」は「または」の意味．すなわち，ファイルオープンに失敗した場合は「die...」以降が実行される．ここで「die...」は状況に応じたエラーメッセージを出力して終了するための決まり文句である．

例題 9: コマンドラインで指定した 1 つ以上のファイルを読み込んで，標準出力（画面）に出力するプログラムを作ろう．存在しないファイルを読もうとした場合にどのようなエラーメッセージが出るか試してみよう．

例題 10: 例題 9 のプログラムを変更して，読み込んだファイル行をソートしてから出力するプログラムを作ろう．

演習のページから，テスト用のテキストファイルを 2 つダウンロードしよう．

```
http://tsujikeita.hp.infoseek.co.jp
        /cje3/sample1.txt
http://tsujikeita.hp.infoseek.co.jp
        /cje3/sample2.txt
```

sample1.txt と sample2.txt の両方を読み込んでソートできているか試そう．結果が正しいかどうか確認するには，Linux の sort コマンドと比較するとよい．sort コマンドの使い方は以下の通りである．

```
% sort sample1.txt sample2.txt
```

5 パターンマッチ

本章はパターンマッチ機能を用いてファイル中の文字列を探索・置換する方法について説明する．

「“perl” というパターンを含むファイル行を探索」「“perl4” を “perl5” に置換」などは簡単な例である．

しかし「英数字だけからなるパターン」や「A で始まり，n で終わるパターン」のような抽象的な指定ができれば便利である．このような融通をきかせるのが「正規表現」というパターン表現方法である．

5.1 正規表現

字面通りの一致: 最も単純な表現である．例えば，

```
ABC567
```

と書けば「ABC567」という文字列にだけマッチする．

文字クラス: アルファベット大文字（1 文字）ならどれでも良いというのは，

```
[A-Z]
```

と書く．ここで，「[]」は範囲を指定するために用いる表記である．アルファベット小文字なら，

```
[a-z]
```

であり，大文字小文字どちらでもよいならば

```
[A-Za-z]
```

と書く．数字ならば，

```
[0-9]
```

であり，もちろん以上をまとめて

```
[A-Za-z0-9]
```

とも出来る．これは「英数字 1 文字にマッチする」．また，

```
[a-ckt-y]
```

は「a 〜 c」「k」「t 〜 y」のどれか 1 文字を意味する．

補集合: ある特定の文字を含んで欲しくない場合は，

```
[^A-Z]
```

のように書く．この例では，アルファベットの大文字「以外」という意味になる．すなわち，アルファベット大文字の「補集合」を表現できる．

繰り返し: 繰り返しを表現したい場合は，

```
[A-Za-z0-9]+     1 回以上繰り返す
[A-Za-z0-9]*     0 回以上繰り返す
[A-Za-z0-9]{N}   丁度 N 回繰り返す
```

と書けばよい．上の 3 つはどれも英数字の繰り返しである．また，以下のように使うこともできる．

```
a+        a, aa, aaa, aaaa, ...
(hoge)+   hoge, hogehoge, hogehogehoge, ...
```

文字列を繰り返したい場合は丸括弧で括る．

特殊な文字: 正規表現には，英数字以外の特殊な文字がある．以下に例を挙げる．

```
\S   空白以外の 1 文字
\s   空白 1 つ
.    任意の 1 文字
```

例えば，

```
\S+\s{3}\S+
```

と書けば，途中に連続した空白 3 文字を含むパターンを表現できる．特殊な文字「そのもの」をマッチさせる場合は，直前にバックスラッシュ「\」をつける．例えば，英数字とピリオドを含むパターンは，

```
[A-Za-z0-9\.]
```

となる．その他の特殊な文字の例を以下に挙げる．

```
+  ?  *  (  )  [  ]  {  }  |  -
```

選択: 「tako」「ika」「fugu」のどれかにマッチさせる場合は以下のように書く．

```
(tako|ika|fugu)
```

位置の表現: 以下の 2 つを比較的よく使う.

- `^` 先頭（補集合とは違うので注意）
- `$` 末尾

例えば「アルファベット小文字で始まり，空白を含まず tion で終わるパターン」は,

```
^[a-z]\S*tion$
```

と書く．末尾に「`$`」を付けると「relationship」のように「tion」以降が続く文字列はマッチしない.

例題 11: 以下を正規表現にしよう.

1. 英数字とピリオドからなる長さ 5 のパターン
2. 数字以外の文字列からなるパターン

5.2 文字列の探索

正規表現で書いたパターンが，文字列`$string`に含まれるかどうか調べるときは，以下のように書く.

```
$string =~ /パターン/
```

等号の後ろにチルダー「`~`」を忘れないように．またパターンはスラッシュ「`/`」で括る．上の式は，もし`$string`が「パターン」を含んでいれば真，含まなければ偽を返す．そこで，以下のように条件分岐に使うことができる.

```
if ($string =~ /^[0-9]/) {
    print "$string\n";
}
```

この例では，`$string`が数字で始まれば，その内容を表示する.

また，パターンに一致した部分だけを抽出する場合は，以下のように指定したパターンを括弧で括る.

```
if ($string =~ /([A-Za-z]+)/) {
    print "$1\n";
}
```

この例では，`$string`中のアルファベット列を抽出する．ここで「`$1`」は特殊な変数であり，抽出された文字列が自動的に代入される．抽出するパターンを複数指定した場合には「`$1`, `$2`, ...」と順番に数字が大きくなる．以下は，どのような結果になるか自分で考えてみよう.

```
if ($x =~ /([A-Z]+)([0-9]+)/) {
    print "$1 $2\n";
}
```

例題 12: 読み込んだファイルから，正規表現で指定したパターンを含む行だけを出力しよう（ファイルの読み込みは 4 章を参照）.

例題 13: 例題 13 を変更して，指定したパターンにマッチした文字列だけを出力しよう.

5.3 文字列の置換

`$string`中の「パターン 1」を「パターン 2」に置換したい場合は以下のように書く.

```
$string =~ s/パターン 1/パターン 2/;
```

スラッシュの前に「`s`」を忘れないように．上の例では，`$string`中の最初のパターン 1 のみが置換される．`$string`中の全てのパターン 1 を置換したい場合は最後に「`g`」を付けて，以下のように書く.

```
$string =~ s/パターン 1/パターン 2/g;
```

また，以下のようにパターン 2 を省略すれば，パターン 1 を削除できる.

```
$string =~ s/パターン 1//g;
```

`$string`中のアルファベットを全て大文字から小文字に変更したい場合は以下のような書き方がある.

```
$string =~ tr/A-Z/a-z/;
```

例題 14: ファイルを読み込んで，指定したパターンを置換しよう.

5.4 正規表現を用いた grep の活用

3.2 節では「`grep`」を用いて条件に合う配列要素だけを抽出した.

```
@x = grep(($_ > 10), @a);
```

この例では，配列`@a`の要素が順番に変数`$_`に渡されて，条件「`$_ > 10`」に合うかどうか評価される．そして条件に合うものだけが配列`@x`の要素となる.

ここでは新たに，条件式に正規表現を使ってみよう．以下の例は，ファイルを読み込んで，`fugu`というパターンを含むファイル行だけを配列`@x`に渡す.

```
chomp(@a = <>);
@x = grep(/fugu/, @a);
```

「`/fugu/`」は「`$_ =~ /fugu/`」と同義である.

6 いろんな処理をしよう

6.1 準備

例題 10 で使用した sample1.txt と sample2.txt を使ってテストしよう.

6.2 行数/単語数のカウント

Linux には，ファイルの行数，単語数，文字数をかぞえるコマンド「wc」がある．ここでは，行数と単語数をかぞえるプログラムを作成し，テストデータを対象に実行しよう．今回のデータは英語なので，空白を手がかりにして単語を抽出することができる．実行結果を「wc」コマンドの結果と比較して，正しくできたか確認しよう．

6.3 複数ファイルへの拡張

「wc」コマンドでは，複数のファイルを与えたときに，各ファイルに関する情報と合計の情報を出力する．そこで，前節のプログラムにも同等の機能を追加しよう．

4 章では「<>」を用いて複数ファイルの内容を読むことを学んだ．しかし，この方法では複数ファイルの内容が 1 つのかたまりとして処理されてしまう．

Perl では，コマンドラインで与えられた引数は配列「@ARGV」に格納される．コマンド名自身は格納されない点に注意しよう．例えば，

```
% perl a.prl aaa bbb ccc
```

を実行すれば，

```
@ARGV = ('aaa', 'bbb', 'ccc');
```

が裏で自動的に実行されると考えればよい．

これを応用して，与えられたファイルを順番にオープンして処理するプログラムの一部を示す．

```
foreach $file (@ARGV) {
    open(IN, $file) || die "$file: $!";
    while ($line = <IN>) {
        chomp($line);
        ...
    }
    close(IN);
}
```

6.4 単語の頻度分布を調べる

データに含まれる単語の頻度を調べて，多い（少ない）順番に出力してみよう．具体的には，まず単語の出現頻度を格納するハッシュを用意する．そして，空白を手がかりに単語を切り出したら，各単語に対応するハッシュ値を増やしていき，最後にハッシュ値に基づいてソートすればよい．

ただし，初期状態ではハッシュ値は必ずしも 0 ではない．そこで安全のため，初出の単語に対しては 1 を代入し，それ以降は「++」を用いて単に増やすようにしよう．出現頻度を格納するハッシュを「%freq」とした場合のプログラムの一部を示す．

```
if (defined($freq{$w})) {
    $freq{$w}++;    # 頻度を 1 つ増やす
} else {
    $freq{$w} = 1; # 初期化
}
```

ここで「defined()」は引数の値が定義されていれば真を，未定義ならば偽を返す組込み関数である．

6.5 接辞処理

「接辞処理（stemming）」とは，活用語や派生語を本来の形（原形）に復元する処理である．例えば，文書検索の索引付けにおいて「apple」と「apples」を異なる索引語と見なすのは都合が悪いので接辞処理が必要である．

今回は，正規表現に基づく文字列置換を用いて簡単な接辞処理を実現しよう．以下に規則の例をいくつか挙げる．各自で規則を追加してほしい．

- 名詞の複数形： 末尾の「s」を削除
- 形容詞の比較/最上級：末尾の「er/est」を削除
- 記号: カンマ，ピリオド，クオート，ハイフンなどを削除
- 大文字と小文字の違い: どちらかに統一
- 動詞から派生した名詞：末尾の「tion」を「te」に置換