日本大学歯学部紀要 32, 85-108, 2004

Perlの活用によるネットワークでのデータ収集

宮崎　洋一[1]　磯川　桂太郎[2,4]　田辺　直紀[1,5]　向山　レイ[3,6]

How to collect data on the network using Perl

Yoichi Miyazaki[1], Keitaro Isokawa[2,4], Naoki Tanabe[1,5] and Rei Mukoyama[3,6]

1. はじめに

Unixにはたくさんの便利なソフトウェアが開発されており, それらをうまく組み合わせれば望みの処理が自動化できる. それがUnixの根本思想でもある. そのためにはスクリプトと呼ばれる簡単なプログラムの作成が必要になる場面が多いが, ほんの少しの努力でスクリプトは書けるようになる. 本論文では, Linuxの動作しているサーバ上でメール・サーバ・プログラムとPerlというコンピュータ言語のスクリプトを連携して, ネットワーク経由でデータ収集を自動的に行う方法を紹介したい.

この方法は本学部の授業『実践医学統計』で実際に活用されており, 身近なデータの統計処理を通して統計の基礎を理解するとともに, 表計算の活用能力を養う, という授業目的を達成する一助となっている.

授業で統計処理の練習として用いる身長などのデータを収集する方法は, いくつか考えられる. 最も原始的で労力と時間を要する方法は, 紙に記入してもらったデータをキーボードから表計算に直接打ち込む方法である. もう少し能率よく行う方法としては, データを電子メールで送信してもらい, 受信したメールを1通ずつ開き, カット・アンド・ペーストで一人一人のデータを表計算に貼り付けていく方法である. 受信メールをテキスト形式で保存するタイプのメール・ソフトならば, カット・アンド・ペーストの単調な作業の部分は, エディタのgrep機能で代用すれば省略可能だ. grep機能とは, 複数のファイルから特定の文字を含んだ行だけを抽出する機能のことである.

これらの方法は, いずれも何らかの形で手作業が必要になってくる. しかし, メール・サーバとPerlスクリプトを連携する本論文の方法ならば, すべてを自動化できる. しかも, WWWサーバと組み合せれば, メール送信されたデータの登録状況をリアル・タイムで確認することができ,

日本大学歯学部　[1]数理情報学教室, [2]解剖学教室第2講座, [3]法医学教室
日本大学歯学部総合歯学研究所　[4]機能形態部門, [5]生体工学研究部門, [6]社会歯学研究部門
〒101-8310　東京都千代田区神田駿河台1-8-13
(受理：2004年11月8日)

[1]Department of Applied Mathematics and Informatics, [2]Department of Anatomy, [3]Department of Legal Medicine, Nihon University School of Dentistry
[4]Division of Functional Morphology, [5]Division of Biomaterials Science, [6]Division of Social Dentistry, Dental Research Center, Nihon University School of Dentistry
1-8-13 Kanda-Surugadai, Chiyoda-ku, Tokyo 101-8310, Japan

収集したデータの配布も容易に行える. データの配布は, WWW ページのデータ・ファイルへのリンクをクリックしてもらうだけで済む.

Unix には PostgresSQL などの本格的なデータベース・ソフトがあるが, 100 人規模のデータではそのようなソフトを使わなくても十分な場合が多い.

本論文は次のように構成されている. 第 2 節では, 身長と靴のサイズのデータ収集を例にとり, データ収集用の Perl スクリプトを使うための設定方法や運用方法を説明する. 第 3 節で Perl に関する基礎知識を解説した後, 第 4 節ではデータ収集用の Perl スクリプトを掲載し, その仕組みを解説する. 第 5 節では, 複数の種類のデータ収集に対応できるように改良したスクリプトを掲載し, 工夫したポイントを述べる.

2. データ収集のための各種設定

この節では, 第 4 節で掲載するデータ収集用のスクリプト `maildata_hf.pl` を使用するために必要な設定について説明する. Unix のシェルの基本操作および DNS, メール,WWW などのネットワークの基礎的な事項を理解していることを前提とする.

説明の便宜のため, メールと WWW のサーバ・プログラムは特定のもの (2.3 項) に限定した. また, プライバシー保護のため, サーバのドメイン名や学生証番号などは架空のものを使った. さらに, 学内 (イントラネット) からのみ接続可能なネットワークを想定しているので, 説明で用いるファイルやディレクトリの属性の設定では, セキュリティ的な面は考慮していない.

2.1. やりたいこと

学生に身長と靴のサイズのデータを電子メールでサーバに送ってもらい, データ・ファイル `hf.txt` に登録する. さらに, データ・ファイルは WWW ページで閲覧または入手できるようにする.

データ・ファイルは以下のように各行が, 学年番号, 氏名, 身長, 靴のサイズ, 受信時刻の 5 項目がコンマで区切られた CSV(Comma Separated Value) 形式のファイルである. ファイルの 1 行目は各項目の内容を表すタイトル行である.

ファイル内容の表示では, 引用の便宜上 1:, 2:のように行番号を示してあるが, 実際はファイルには書かれていない.

データ・ファイル `hf.txt`

```
1:  number,name,height,feet,time
2:
3:  002,東　京子,155.7,23.0,10-27 15:16:44
4:  004,岡地　真知子,173.0,27.5,10-27 15:23:27
5:  005,御茶ノ水　博士,169.0,26.5,10-27 15:24:20
6:  006,神田　明,181.0,27.0,10-27 15:22:19
7:  008,品川　次郎,174.5,27.0,10-27 15:23:24
```

2.2. 準備するファイル

次の 2 つのファイルを準備する.

- `maildata_hf.pl`　第 4 節に掲載してある Perl スクリプト

- `std_list.txt`　学生名簿ファイル

`maildata_hf.pl` がこのデータ収集において中心的役割を果たすファイルである.

学生名簿ファイルは, 各行が学年番号, 学生証番号, 氏名, ふりがな, 性別の項目からなる CSV 形式のファイルである. なお, 学生に付与されている番号には, 入学から卒業まで有効な学生証番号とその年度だけで有効な学年番号の 2 通りあり, 授業では通常, 学年番号が使用されている. データ収集で実際に使用されるのは, 学年番号, 学生証番号, 氏名の 3 項目である.

学生名簿ファイル `std_list.txt`

```
 1:  1,SD04002,秋葉　太郎,あきば　たろう,男
 2:  2,SD04003,東　京子,あづま　きょうこ,女
 3:  3,SD04005,上野　公夫,うえの　きみお,男
 4:  4,SD04009,岡地　真知子,おかち　まちこ,女
 5:  5,SD03015,御茶ノ水　博士,おちゃのみず　ひろし,男
 6:  6,SD04010,神田　明,かんだ　あきら,男
 7:  7,SD03021,静岡　静香,しずおか　しずか,女
 8:  8,SD04014,品川　次郎,しながわ　じろう,男
 9:  9,SD04015,渋谷　四朗,しぶや　しろう,男
10: 10,SD03024,目黒　五郎,めぐろ　ごろう,男
```

2.3. サーバの動作環境

データ収集用のサーバ (1 台) のホスト名は `mackerel` とし, 次の条件を満たしているものとする.

- OS として Linux[1] (Plamo Linux[2] など) がインストールされている.
- ドメイン名は `mackerel.edu.private-net` である.
- メール・サーバ・プログラム `qmail`[3] (Bernstein 作) が標準設定でインストールされている.
- このサーバに登録されたローカル・ユーザ *local-user* へのメールは

  local-user`@mackerel.edu.private-net`

  というメール・アドレスで届く.
- WWW サーバ・プログラム `apache`[4] がインストールされている. WWW サーバのドキュメント・ルートは `/htdoc` であり, `/htdoc` 直下のファイル `htdoc/`*html_file* はブラウザで

  `http://mackerel.edu.private-net/`*html_file*

  にアクセスすることでクライアント PC から閲覧できる.

2.4. ユーザ kyoin の登録とホーム・ディレクトリの設定

まず, `root` ユーザになってサーバ `mackerel` にユーザ `kyoin` を登録しておく. `kyoin` のホーム・ディレクトリは`/home/kyoin` とする. 次にサーバへユーザ名 `kyoin` で `login` して次の設定を行う. 具体的なコマンドはコマンド・プロンプト`%` の後に示してある.

- 2 つのディレクトリ `/home/kyoin/tokei`, `/home/kyoin/tokei/tmp` を作成し, これらのディレクトリ属性を読込可・書込可・実行可とする.

  ```
  % mkdir tokei
  ```

```
% mkdir tokei/tmp
% chmod a+rwx tokei tokei/tmp
```

- データ・ファイル hf.txt と一時ファイル tmp/tmp-hf.txt を作成し, これらのファイル属性を読込可・書込可にする.

```
% touch hf.txt tmp/tmp-hf.txt
% chmod a+rw hf.txt tmp/tmp-hf.txt
```

- 2.2 項で準備した Perl スクリプト maildata-hf.pl を/home/kyoin/tokei に置き, そのファイル属性を読込可・実行可にする.

```
% chmod a+rx tokei
```

2.5. サーバの設定

root ユーザになって次のコマンドを実行する. 具体的なコマンドはコマンド・プロンプト# の後に示してある.

- メール・エイリアスを設定して, メール・アドレス tokei-en@edu.private-net 宛のメールが Perl スクリプト maildata_hf.pl に渡るようにする.

```
# cd /var/qmail/aliase
# echo | /home/kyoin/tokei/maildata.pl > .qmail-tokei-en
```

- /home/kyoin/tokei/直下のファイル/home/kyoin/tokei/*file* がブラウザで

  http://mackerel.edu.private-net/tokei/*file*

  にアクセスすることでクライアント PC から閲覧できるようにする.

```
# ln -s /home/kyoin/tokei /htdoc
```

注： メール・エイリアスから呼び出される Perl スクリプトは, qmail のインストール時に登録される aliase ユーザの権限で実行される.

2.6. 学生への指示

学生へは次の操作を指示する.

- 身長と靴のサイズのデータを次のようにして電子メールで送信する.
  1) 送信先 tokei-en@mackerel.edu.private-net
  2) 題名　height
  3) メール本文　一行目に次の 6 項目をコンマ区切りで書く

第 1 項目	data:height	第 4 項目	自分の身長 (cm)
第 2 項目	自分の学年番号（3 桁で）	第 5 項目	自分の靴のサイズ (cm)
第 3 項目	自分の学生証番号	第 6 項目	end

  例　data:height,004,SD04005,174.3,26.5,end

- 送信したデータが正しく登録されたかをブラウザで

  http://mackerel.edu.private-net/tokei/hf.txt

にアクセスして確認する．データを修正したいときは，上の書式に則って再度送信すればよい．

- 全員のデータ送信が終了したら，データ・ファイルを表計算に取り込む．具体的な操作としては，ブラウザに表示されたデータをコピーして，CSV形式のデータとして表計算のシートに貼り付ければよい．この操作によりデータの各項目が表計算の各列に取り込まれる．

2.7. 動作チェック

正しい書式のデータが電子メールで送られ，Perlスクリプト `maildata_hf.pl` が実行されると，データ・ファイルを作成する前段階として，データは一時ファイルに氏名および受信時刻とともに追記されていく．この一時ファイルから各学生について最新のデータを抽出し，学年番号順に並べ替えたものがデータ・ファイルに書き込まれる．例えば，下の一時ファイル `tmp-hf.txt` からは，2.1項で示した5人分のデータ・ファイルが生成される．2つのファイルを比較してわかるように，「岡地　真知子」と「御茶ノ水　博士」はデータを2回以上送信しているが，データ・ファイルには最後に送ったもののみが反映されることになる．

一時ファイル `tmp-hf.txt`

```
1:  004,岡地　真知子,142,21.5,10-27 15:14:19
2:  002,東　京子,155.7,23.0,10-27 15:16:44
3:  005,御茶ノ水　博士,171.0,26.5,10-27 15:15:19
4:  005,御茶ノ水　博士,169.9,26.5,10-27 15:21:38
5:  006,神田　明,181.0,27.0,10-27 15:22:19
6:  008,品川　次郎,174.5,27.0,10-27 15:23:24
7:  004,岡地　真知子,173.0,27.5,10-27 15:23:27
8:  005,御茶ノ水　博士,169.0,26.5,10-27 15:24:20
```

この仕組みを念頭において，データ収集システムの動作チェックを次のように行う．

- まず，サーバにユーザ名 `kyoin` で `login` して，`echo` コマンドと組み合わせてPerlスクリプトを次のように実行する．

```
% cd /home/kyoin/tokei
% echo data:height,001,SD04002,155,22,end | ./maildata_hf.pl
```

正しく設定されていれば，2つのファイル `tmp-hf.txt`, `hf.txt` に

```
001,秋葉　太郎,155,22,10-25 15:26:38
```

のような行が書き込まれているはずである．`echo` で使う学年番号の `001` と学生証番号の `SD04002` は，学生名簿ファイルにあるものと一致する組み合わせでなければならない．

ファイルの読込または書込に失敗した場合は，`"Cannot open tmp file"` などのエラーメッセージが表示されるので，ファイル属性の設定に誤りがないかを確認する．

- 次に，電子メールで以下のようなデータを送り，正しくファイルに記録されるかを確認する．

```
data:height,001,SD04002,155,22,end
```

3. Perlの基礎

Perlは文字列や数値などのデータの処理を得意とするコンピュータ言語であり, WWWサーバに設置されるCGI(Common Gateway Interface)を記述するプログラム言語の一つとして活用されている. 汎用性のあるC言語と比べると, 手軽にスクリプト(プログラム)を書けるのがよい. C言語で5行必要なプログラムがPerlでは1行で済むことがある. C言語では, 変数が整数型なのか文字型なのかの宣言をしてから使わなければならないのに対し, Perlではそうした宣言をせずに新しい変数を使用できる. 同一の変数でもある時は文字型, ある時は整数型というように, 状況に応じて柔軟に使い分けが可能なのもPerlの特徴の一つである.

少し大胆な表現をすれば, Perlは, いわゆる「ぼくはうなぎだ」文[5]をそのときの状況に応じて正しく解釈し実行してくれる. つまり, 「ぼくはうなぎだ」という文が小説の冒頭にあれば, 「夏目漱石の『吾輩は猫である』のように, うなぎの視点から見た人間世界を描写する小説がこれから始まる」のだと解釈してくれるし, 「みんなはどんな食べ物が好きですか?」の問いに対して, 「わたしは鰻」, 「ぼくはうなぎだ」と答えている場面だとすれば, 「ぼくの好きな食べ物はうなぎだ」と解釈してくれる.

Perlにはたくさんの参考書がある. 『らくだ本』の通称で知られる, Perlの作者L.Wall自らが著した本[6]が有名であり, バイブル的な存在であるが, 初心者には敷居が高すぎる. 標準的な入門書[7]と手元に置いて参照用に使える本[8]を参考文献の欄に挙げておく.

著者の一人がPerlを使うきっかけとなった参考書は『入門Perl』[9]である. この本はUnixサーバ管理者のためのシリーズの一冊で, 前半の入門的な部分はPerlの活用方法が説明されており, 非常に示唆に富むものであった. とくに, CSV形式のファイルから高級な組版ソフトであるTEXのソース・コードを自動的に作成する方法は印象的であった.

この節では, Perlについてゼロから出発して, 第4節と第5節で紹介するPerlスクリプトを理解し作成できるレベルまでの内容を解説する. 本節の説明を十分に理解すれば, 別の用途でもPerlを活用していけるはずである.

3.1. Perlの実行

以下の操作は`kterm`などのUnix端末で行っているものとし, コマンド・プロンプトは%で表すことにする. 端末画面で%のついてない行はコマンドを実行した結果, 画面に表示される行である.

Perlスクリプトを実行する手順は, スクリプト・ファイル`ex-prn.pl`を例にとると次のようになる.

1) スクリプト・ファイル`ex-prn.pl`をエディタで編集する.
2) `ex-prn.pl`のファイル属性を実行許可モードにする.

```
% chmod a+x ex-prn.pl
```

3) Perlによりスクリプト`ex-prn.pl`を実行する.

```
% ./ex-prn.pl
```

下に`ex-prn.pl`のファイルの内容とその実行結果および説明を挙げる.

スクリプト ex-prn.pl

```
1: #!/usr/bin/perl
2: print "Conclusion:\n I have been becoming wealthy ";  # comment (ignored)
3: print
4: "since I began to learn Perl.\n";
```

端末画面

```
% ./ex-prn.pl
Conclusion:
 I have been becoming wealthy since I began to learn Perl.
```

- 1行目はPerlプログラム本体を呼び出すための命令でどんなスクリプトも必ずこの行で始める. #!に続けてPerlプログラム本体を絶対パスで書いておく. Perlプログラム本体の絶対パスは次のコマンドで調べることができる.

```
% which perl
```

- 2行目以降が実際の命令になっている.
- 命令文の終わりには必ずセミコロンを付ける. 2行目のように命令を1行で書いてもいいし, 3-4行目のように2行にわたって書いてもよい.
- print関数は, ダブルクォーテーションで囲まれた文字列を表示させる命令である.
- \nは改行コードを意味する. 2行目でConclusion:の直後に\nがあるので, ここで改行されている.
- 各行において, #以降は無視されるので, コメントなどを書いておける. ただし, 1行目の#は例外である.

急ぎのときは上述の手順の2) と3) の部分を次で代用できる.

```
% perl ex-prn.pl
```

この場合は, 1行目の#!/usr/bin/perlは書かなくてもよい.

3.2. 変数と演算

スクリプト ex-var.pl

```
1:  #!/usr/bin/perl
2:  $birthday=1970;
3:  $thisyear=2004;
4:  $age=$thisyear-$birthday;
5:  $name="Jack";
6:  print "$name is $age years old.\n";
7:  $str=$birthday.$thisyear;
8:  print "$str\n";
```

端末画面

```
% ./ex-var.pl
Jack is 34 years olds.
19702004
```

- $age などのように$で始まる文字列は Perl における変数 (より正確にはスカラー型の変数) を表す.
- 2 行目や 5 行目のように等号 = を使うことで, 左辺の変数に右辺 (文字列や数値) を代入することができる. 文字列はダブルクォーテーションで囲む必要がある.
- 数値どうしの加減乗除 + − ×÷ を行うには, それぞれ + - * / を使う. 3 行目では $thisyear の値から$birthday の値を引いた結果を$age に代入している.
- ピリオド. は文字列どうしを連結する演算子である. 7 行目では 1970 と 2004 を文字列として連結させた結果の 19702004 を変数$str に代入している.
- 6 行目の print 文に 2 つの変数$name, $age があるが, 出力されるのは変数名そのものではなく変数の値である.

その他よく使われるものを挙げておく.

書 式	説 明
$a=$a+1	等式 = は, 右辺の計算結果を左辺に代入することを意味するので, 変数$a の値を 1 増やすことと同じ.
$a++	$a=$a+1 の簡略な書き方.
$a.="*strings*"	$a=$a."*strings*"; と同じ. つまり, 変数$a に文字列 *strings* を連結させる.

3.3. if 文

if 文は, 次のような書式になっていて, 与えられた条件 *condition* の真偽を判定して, 真のときは *command1* を実行し, 偽のときは *command2* を実行する.

書式： if(*condition*){*command1*;}else{*command2*;}

また, 次の書式のように else 以下は省略することが可能で, その場合は条件 *condition* が真のときに *command1* を実行し, 偽のときは何もしない.

書式： if(*condition*){*command1*;}

スクリプト ex-if.pl

```
1:  #!/usr/bin/perl
2:  $a=10;
3:  if($a == 10){print "a equals 10.\n";}
4:  else{print "a does not equal 10.\n";}
5:  $b=8;
6:  if($b > 10){print "b is greater than 10.\n";}
7:  else{print "b does not exceed 10.\n";}
```

端末画面

```
% ./ex-if.pl
a equals 10.
b does not exceed 10.
```

- 3-4 行目では変数$a の値と 10 が等しいかを判定して, その真偽により表示する内容を変え

ている. 2つの数値が等しいかどうかを比較する場合は, 2つの等号==で行う. 等号が一つだけだと代入になってしまうので注意が必要である.

- 6-7行目では変数$bの値が10よりも大きいかを判定して, その真偽により表示する内容を変えている.

3.4. 条件

if文でよく使われる条件を挙げておく.

条　件	意　味
`$a == $b`	変数$aと変数$bを数値として比較し, 等しいとき真を返す.
`$a != $b`	変数$aと変数$bを数値として比較し, 等しくないとき真を返す.
`$a eq $b`	変数$aと変数$bを文字列として比較し, 一致するとき真を返す.
`$a ne $b`	変数$aと変数$bを文字列として比較し, 一致しないとき真を返す.
`!(`*condition*`)`	条件 *condition* の否定が真のとき, つまり,*condition* が偽のとき真を返す.

3.5. 特殊変数 $_

Perlにはいくつかの特殊変数が用意されている. それらのうち変数$_はよく使われ, 3.12項で詳述するように, ファイルの読込において1行ずつ読み込まれた内容は変数$_に代入される. この変数は省略可能なためスクリプト中に現れることは稀である. 例えば, 単に

```
print;
```

とした場合, print関数の後に何も書かれていないが, これは

```
print $_;
```

と同じ意味である. すなわち, 関数の引数が省略されたときは変数$_がその関数の引数となる.

3.6. 文字列のパターン

Perlでは, 文字列の中に特定のパターンの文字列があるかを判定したり, 特定のパターンの文字列を別のパターンの文字列で置き換えることができる.

書　式	意　味
`$a=~ m/`*string*`/`	変数$aが文字列 *string* を含んでいるとき真を返す.
`/`*string*`/`	`$_=~ m/`*string*`/` と同じ.
`$a=~ s/`*string1*`/`*string2*`/`	変数$aの中で最初に現れた *string1* というパターンの文字列を *string2* に置き換える.
`$a=~ s/`*string1*`/`*string2*`/g`	変数$aの中で *string1* というパターンの文字列をすべて *string2* に置き換える. オプションのgはglobal(全体にわたって)の意.
`s/`*string1*`/`*string2*`/g`	`$_=~ s/`*string1*`/`*string2*`/g` と同じ.
`[0-9]`	0から9の数値のどれかを表す.

スクリプト ex-pattern.pl

```
 1:  #!/usr/bin/perl
 2:  $a="Jack Chack";
 3:  $a=~ s/ack/oke/;
 4:  print "$a\n";
 5:  $a="Jack Chack";
 6:  $a=~ s/ack/oke/g;
 7:  print "$a\n";
 8:  $b="Betty01";
 9:  if($b=~ m/[0-9][0-9]/){print "$b matches the pattern.\n";}
10:    else{print "b does not match the pattern.\n";}
11:  $c="Be0tty1";
12:  if($c=~ m/[0-9][0-9]/){print "$c matches the pattern.\n";}
13:    else{print "c does not match the pattern.\n";}
```

端末画面

```
% ./ex-pattern.pl
Joke Chack
Joke Choke
Betty01 matches the pattern.
Be0tty1 does not match the pattern.
```

- 変数$a には"ack"というパターンが 2 箇所ある. 3 行目では g オプションが付かないため, Jack が Joke と置換されるだけで, Chack のほうは変更を受けない. それに対し, 6 行目では g オプションがあるので, Chack も置換されて Choke になる.
- 8-13 行目では変数$b と$c について, 2 つ連続した数字の並びがあるかを調べている. 変数$b はこのパターンを含んでいるが, 変数$c は含んでいない.

その他の文字列のパターンを次に挙げておく.

書　式	意　味
^	文字列の先頭を表す記号.
$	文字列の末尾を表す記号.
$a=~ m/^height/	変数$a が height で始まっているとき真を返す条件.

3.7. chop

書式　chop(*string*)　　意味: 文字列 *string* から最後の文字を切り落とす.

chop 関数は, ファイルを 1 行ずつ読み込むときに, 行末にある改行コードを切り落とすのに使われることが多い. この目的で使うなら, 改行コードのみを切り取る chomp のほうが安全なようだ.

3.8. for 文

処理の繰り返し (ループ) には for 文や while 文が使われる. 次の書式は for 文の最も基本的な使い方である. 変数$i に 1 から 5 までの整数を順次代入して, *command* を繰り返し実行するという意味である.

書式 `for $i (1..5){`*command*`;}`

スクリプト ex-for.pl

```
1:  #!/usr/bin/perl
2:  $a="Jack";
3:  for $i (1..5){
4:      print"$i -> $a\n";
5:      chop($a);
6:  }
```

端末画面

```
% ./ex-for.pl
1 -> Jack
2 -> Jac
3 -> Ja
4 -> J
5 ->
```

ex-for.pl では, 変数$i に 1 から 5 までの整数を順次代入して, 4 行目と 5 行目の命令が繰り返し行われる. 5 行目の chop は変数$a に代入された文字列の最後の文字を切り落とす関数なので, $i=1 のときの処理が終わった段階で, Jack の最後の文字 k が切り落とされて, $a="Jac" となり, $i=2 の処理に入る. 以下同様に$i が 1 増えるごとに$a の末尾の文字が 1 つずつ切り落とされていく. その結果, 出力は上のようになる.

3.9. 配列

数学における数列 $a_0, a_1, \cdots$ と同様に perl では次のように添字番号付きの変数

`$mon[0], $mon[1], $mon[2], $mon[3], . . .`

を扱うことができる. これを配列と呼ぶ. 添字番号は 0 から始まる整数である. 配列の各要素は $mon[0] のように$で始め, 添字番号は [ ] で囲む. 配列全体を表すには, @mon のように配列名を表す文字列 mon の先頭に@を付ける.

配列に関係した関数には次のようなものがある.

関　数	説　明
split(/*separator*/,$s)	変数$s を *separator* で区切って配列を作る.
join(/*connector*/,@array)	配列@array の各要素を *connector* で連結して文字列を作る.
pop(@array)	配列@array の最後の要素を削除する.
	$l=pop(@array) となっていれば, 切り取られた最後の要素は変数$l に代入される.

スクリプト ex-array.pl

```
1:  #!/usr/bin/perl
2:  $mon[0]="January";
3:  $mon[1]="February";
4:  $mon[2]="March";
5:  $mon[3]="April";
```

```
 6:  for $i (0..3){print "$i -> $mon[$i]\n";}
 7:  $s="May,June,July";
 8:  @month=split(/,/,$s);
 9:  for $i (0..2){print "$i -> $month[$i]\n";}
10:  $l=pop(@month);
11:  print "pop -> $l\n";
12:  for $i (0..2){print "$i -> $month[$i]\n";}
13:  $t=join(/:/,@mon);
14:  print "t -> $t\n";
15:  $size=@mon;
16:  print "number of mon = $size\n";
```

端末画面

```
% ./ex-array.pl
0 -> January
1 -> February
2 -> March
3 -> April
0 -> May
1 -> June
2 -> July
pop -> July
0 -> May
1 -> June
2 ->
t -> January:February:March:April
number of mon = 4
```

- 2-5 行目では, 配列@mon を各要素に直接代入することによって定義している. 6 行目では, 配列@mon の各要素を添字番号とともに表示させている.
- 8 行目では, split 関数を使ってコンマで区切られた文字列 (変数$s) から配列@month を作っている. 9 行目では, 配列@month の各要素を添字番号とともに表示させている.
- 10 行目では, pop 関数を使って配列@month から末尾の要素 July を削除し, 取り除いた July を変数$l に代入している. 11 行目で変数$l を表示させている. 12 行目で再度@month の各要素を添字番号とともに表示させているが, $month[2] が pop 関数で削除されているため, 添字番号 2 に対応する要素は空になっている.
- 13 行目では, @mon の要素をセミコロンで連結して文字列を作り, それを変数$t に代入し, 14 行目で変数$t を表示させている.
- 15 行目は, 等号の左辺が文字列や数値を扱うスカラー変数なのに対して, 右辺が配列変数である. 変数の型が違うので, 通常はこのような代入は行えないが, この場合は例外で, 配列@mon の大きさ (要素の個数) が変数$size に代入される. 添字番号が 0 から始まっているので, 配列の大きさは末尾の要素の添字番号よりも 1 大きい.
  この使い方からわかるように, Perl は「象の鼻はキリンよりも長い」的な文を適切に解釈してくれる. したがって, Perl は「象の鼻はキリンのそれよりも長い」と表現しなければならない英語よりも日本語に近い言語なのかもしれない.

配列から変数や別の配列を定義するには, 次の方法が便利である. split 関数と組み合わせれば, CSV 形式のデータの各項目を変数に代入する操作も簡単にできる.

書　式	説　明
($a,$b)=@array	$a=$array[0] と$b=$array[1] と同じ.
($a,$b,@c)=@array	$a=$array[0], $b=$array[1] として, しかも, @array の 2 番目以降の要素$array[2], $array[3], $array[4], . . . からなる配列を@c とすることと同じ.

3.10. ハッシュ（連想配列）

添字番号に値 (数値または文字列) を対応させるのが配列だとすれば, ハッシュはキーに値を対応させるものと表現できる. つまり, 配列ではキーに相当するのが添字番号であったが, ハッシュではキーとして数値, 文字列のどちらも使用できる.

%days のように%で始まるものがハッシュを表す. 各キーに対応する値を表すには, $days{*key*} のようにハッシュ名の先頭に$を付け, キー *key* を { } で囲む. 例えば, %days を各月の英語名にその月の日数を対応させるハッシュとすれば, 1 月の日数は$days{"January"} で表される.

スクリプト ex-hash.pl

```
1:  #!/usr/bin/perl
2:  $days{"January"}=31;  $days{"February"}=28;
3:  $days{"March"}=31;    $days{"April"}=30;
4:  for $i (sort keys %days){
5:      print "$i -> %days\n";
6:  }
```

端末画面

```
% ./ex-hash.pl
April -> 30
February -> 28
January -> 31
March -> 31
```

- 2-3 行目で, 1 月から 4 月の各月に日数を対応させるハッシュ%days を定義している.
- 4-6 行目では, ハッシュ%days のキーをアルファベット順に並べて, 月名 (キー) と日数 (値) を表示させている. 4 行目の keys はハッシュのキーからなる配列を作る関数であり, sort はアルファベット順に並べ替える関数である.

配列と同様に, 定義されていないキーに対するハッシュの値は空文字または 0 とみなされる.

キーを添字番号とする限りにおいては, 配列とハッシュの使い方に大差はない. しかし, 次の点を注意する必要がある. 配列では添字番号を 0 から始めるため, $a[10001] から$a[10100] まで範囲だけが必要な場合でも, 自動的に$a[0], $a[1], $a[2], . . . ,$a[10000] の 10001 個分の格納領域がメモリー上に確保される. そのためメモリの無駄が発生してしまう. このようなときはハッシュを使うのが効率的である.

3.11. while 文

while 関数は for 文と並んで, 処理の繰り返しでよく使われる. 次の書式のように, while 関数は *condition* が真である間, *command* を繰り返す.

書式 while(*condition*){*command*;}

ファイルなどの読込の際には, 次節で示すようにファイル・ハンドルとともに

書式 while(<*file_handle*>){*command*;}

という形で使用される. これにより, *file_handle* を通してファイルなどを 1 行ずつ読み込んで *command* を実行するという操作が, ファイルの最終行まで繰り返される.

3.12. ファイル操作

ファイルの読込と書込は, open 関数でファイルを開くことで行う. 直接ファイルとやりとりするのではなく, ファイルハンドルを通して行われる. ファイルに対する処理が終了したら, close 関数でファイルを閉じておく.

書　式	説　明
open(F,"*file*")	ファイル・ハンドルを F に設定し, *file* を読込モードで開く.
open(F,">*file*")	*file* を書込モードで開く. 既にあるファイルの内容は上書きされる.
open(F,">>*file*")	*file* を書込モードで開く. 既にあるファイルの内容に追記していく.
close(F)	ファイル・ハンドル F で開いたファイルを閉じる.

ファイルが存在しない, 書込が禁止されているなどの理由でファイルのオープンに失敗する可能性に備えて, open 関数は次の書式で die 関数と一緒に使い, ファイルのオープンに失敗したら, エラー・メッセージを表示させてプログラムを終了するようにしておく.

open(F,"*file*") || die "*error_message*.\n";

スクリプト ex-fopen.pl

```
1:  #!/usr/bin/perl
2:  open(L,"ex-list.txt") || die "Cannot open ex-list.txt\n";
3:  while(<L>){
4:      chop;
5:      ($num,$gnum,$name)=split(/,/);
6:      print "$num,$name\n";
7:  }
```

スクリプト ex-list.txt

```
1:  1,SD04002,秋葉　太郎,あきば　たろう,男
2:  2,SD04003,東　京子,あづま　きょうこ,女
3:  3,SD04005,上野　公夫,うえの　きみお,男
```

端末画面

```
% ./ex-fopen.pl
1,秋葉　太郎
2,東　京子
3,上野　公夫
```

- 2行目でファイル ex-list.txt を読込モードで開き, ファイル・ハンドルを L に設定している.
- 3-7行目では, ファイル・ハンドル L を通してファイル ex-list.txt を1行ずつ読み込んで, 4-6行目の処理を繰り返している.
- 4行目では, 行末の改行コードを切り落としている.
- 5行目では, split 関数を使って, 3つの変数$num, $gnum, $name への代入を行っている. ファイル ex-list.txt の各行は, 学年番号, 学生証番号, 氏名, ふりがな, 性別がコンマ区切りのデータとなっているので, $num, $gnum, $name にはそれぞれ, 学年番号, 学生証番号, 氏名が代入される.
- 6行目で, 学年番号と氏名を表示させている.

ex-fopen.pl ではファイルからの読込であったが, while(<>) のようにファイル・ハンドルに何も書かないときは, 標準入力からのデータの読込になる. 標準入力とはパイプ | を通して渡されたデータなどを指す.

下の例は, Unix シェルの cat コマンドを使ってファイル ex-list.txt を読み込み, そのデータをパイプを通して, Perl プログラム ex-stdin.pl に渡して処理させている. したがって, 上の例と同じ出力結果が得られる.

スクリプト ex-stdin.pl

```
1:  #!/usr/bin/perl
2:  while(<>){
3:      chop;
4:      ($num,$gnum,$name)=split(/,/);
5:      print "$num,$name\n";
6:  }
```

端末画面

```
% cat ex-list.txt | ./std-in.pl
1,秋葉　太郎
2,東　京子
3,上野　公夫
```

3.13. 時間

localtime 関数により, Perl を実行しているマシンに設定された現在のローカル時刻が次のような配列として取得できる.

0. 秒　1. 分　2. 時　3. 日　4. 月　5. 年　6. 曜日　7. 年日　8. 他

曜日は日曜日, 月曜日, ··· に対して 0, 1, ··· が対応する. 月は1月が 0, 2月が 1 というように 1 ずれているので注意が必要である. また, 年は「西暦 −1900」が返るので, 西暦を求めるにはその値に 1900 を加えればよい.

マシンが日本時間に設定されていれば, ローカル時刻は世界標準時プラス 9 時間である.

スクリプト ex-time.pl

```
1: #!/usr/bin/perl
2: ($sec,$min,$hour,$day,$mon,$year,$wday,$tday)=localtime;
3: print "$sec,$min,$hour,$day,$mon,$year,$wday,$tday\n";
```

端末画面

```
% ./ex-time.pl
7,23,19,28,9,104,4,301
```

ex-time.pl の実行例では, $1900 + 104 = 2004$ と $9 + 1 = 10$ より, 現在時刻が 2004 年 10 月 28 日 (木) 19 時 23 分 7 秒 であり, 元旦から数えて 301 日目であることがわかる.

3.14. 日本語を扱うときの注意

コンピュータで扱う日本語の文字コードには, JIS, シフト JIS, EUC の 3 種類があり, Perl で日本語を扱うときは注意が必要である. Unix で Perl を動かすときは, 日本語を含むテキスト・ファイルの文字コードを EUC(Extended Unix Code) にしておけば問題が起こらない. スクリプトが文法的に正しくても意図通りに動作しない場合, 日本語の文字コードが原因になっていることが多い.

Unix で使える文字コードの変換プログラムには, nkf や qkc(Kimihiko Sato 作) などがある. qkc は次の「超高速漢字コード変換プログラム」のページ [10] から入手し, インストールすればよい. 日本語のファイルの文字コードを EUC にすると同時に改行コードを UNIX 形式のものに変換するには, 次のコマンドを実行する.

```
% qkc -e -u file_name
```

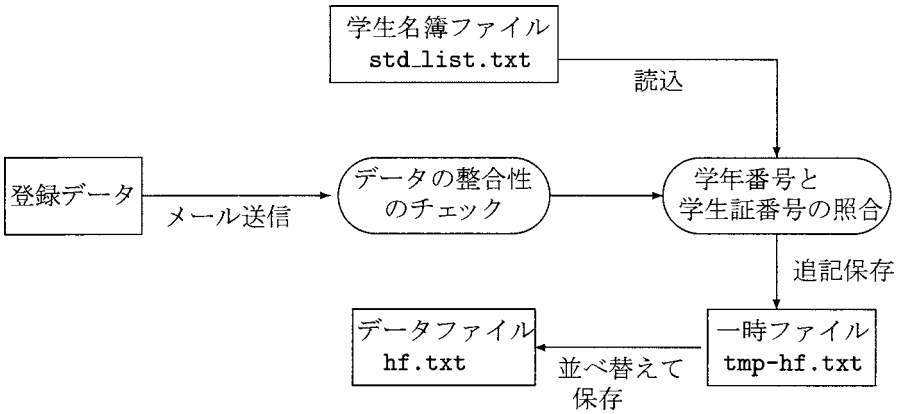
3.15. その他

第 4 節と第 5 節で必要になるその他の関数を挙げておく.

関　数	説　明
exit	プログラムを終了する.
chdir(*dir*)	Perl が動作するときのディレクトリ *dir* に設定する.
length(*strings*)	文字列 *strings* の長さを求める関数.
last	ループから抜ける.
lc(*strings*)	文字列 *strings* の中のアルファベット文字を小文字 (lower case) に変換する関数.

4. データ収集用スクリプト

この節では, 以下で掲載する Perl スクリプト `maildata_hf.pl` の仕組みについて説明する. このスクリプトにより, 電子メールで送信された身長と靴のサイズのデータは処理され, データ・ファイル `hf.txt` に保存される. データの流れは下図のようになっている.



送信された電子メールは, `data:height` と書かれた行 (2.6 項参照) だけが抽出され, データの整合性のチェックを受けた後, 学生名簿ファイルの読込により学年番号と学生証番号が一致するかの照合を受ける. これをクリアしたデータは一時ファイル `tmp-ht.txt` に保存される. その後, 一時ファイルを元にして, 学年番号順に並べ替えが行われ, 最終的にデータ・ファイル `hf.txt` が作成される. 1 人の学生が修正などのためにデータを複数回送信してきた場合は, 最後に送信されたものだけがデータ・ファイルに記録される.

`maildata_hf.pl` の処理は次の 7 つのステップからなる.

1. 作業ディレクトリ, ファイル名, データのタイトル行, データ項目の個数を各変数に設定する.
2. メールを読み込んで, データ行について文字列 `data:height` が含まれる行を抽出し, データの整合性をチェックする. 同時に受信時刻を記録する.
3. 学生名簿ファイルを読み込む.
4. 学年番号と学生証番号を照合する.
5. 学年番号, 氏名, データ (身長と靴のサイズ), 受信時刻を一時ファイルに保存する.
6. 一時ファイルのデータを学年番号順に並べ替える.
7. 並べ替えたデータをデータ・ファイルに書き込む.

`maildata_hf.txt` の各行の意味はスクリプト中にコメントとして説明してあるので, ここでは, いくつかの注意点と補足を述べておく.

- 後で変更になる可能性のあるものは, スクリプトの始めのほうに集めておくのがよい. 未来の自分や他人が, 使用環境に合わせてスクリプトを変更しやすくするためである. この

スクリプトで変更になる可能性があるものは, 3-9 行目のファイル名やデータ項目の個数などである.

例えば, データ・ファイル hf.txt の処理に関わるのは

8 行目　$data_file="hf.txt";

62 行目　open(F,"> $data_file") || die "Cannot open data file\n";

であるが, 変数$data_file を使わずに, 8 行目を削除して, 62 行目を

open(F,"> hf.txt") || die "Cannot open data file\n";

で置き換えても実行結果は全く同じである. しかし, そうするとデータ・ファイル名を別なファイル名で置き換える必要が生じたときに不便である. スクリプト中のデータ・ファイルが関係している箇所を隅から隅まで探し回って修正しなければならないからだ. 変数を使ってスクリプトの始めのほうで定義しておけば, このような不便は起こらない.

- 15 行目で変数$in_num に代入される学年番号は, 001 のように 3 つの数字からなる文字列であり, 1 という数値ではない. しかし, 41 行目では配列@gnum の添字番号として $gnum[$in_num] のように使われている. つまり, $in_num に 001 が代入されていたとしても, Perl は$gnum[$in_num] を$gnum[001] ではなく$gnum[1] と解釈してくれるので文法エラーにはならない. このように Perl では文字列として定義された変数でも, 呼び出される状況に応じて適切に解釈してくれる. この柔軟性が, 第 3 節の冒頭で「ぼくはうなぎだ」文を引き合いに出して絶賛した Perl の長所の一つである.
- 28 行目の""は, ダブルクォーテーションで何もはさんでいないので, 空 (から) の文字列を意味する. したがって, 「$in_data eq ""」は「変数$in_data が空」という条件になる.

スクリプト maildata_hf.pl

```
 1: #!/usr/bin/perl
 2: ############################## Step1 変数の設定
 3: $workdir="/home/kyoin/tokei";                  # 作業 directory
 4: $std_list="std_list.txt";                      # 学生名簿ファイル
 5:
 6: $tmp_file="tmp-hf.txt";                       # 一時ファイル
 7: $data_item="number,name,height,feet,time";     # データのタイトル行
 8: $data_file="hf.txt";                           # データ・ファイル
 9: $dnum=2;                                      # データ項目の個数
10: ############################## Step2 メール・データの整合性チェック
11: while(<>){
12:     chop;                                     # 行末の改行コードの削除
13:     s/ //g;                                   # 不要なスペースの削除
14:     if(/^data:height/){                       # 行頭に data:height があるか
15:        ($header,$in_num,$in_gnum,@mdata)=split(/,/);  # 各変数へ代入
16:        $end=pop(@mdata);                      # @mdata の末尾を切り取り左辺に代入
17:        if($end ne "end"){exit;}               # 最後の項目が end か
18:        if(@mdata != $dnum){exit;}             # データ項目の個数が合っているか
19:        if(length($in_num) != 3){exit;}        # 学年番号は 3 桁か
20:        if(!($in_num=~ m/[0-9][0-9][0-9]/)){exit;} # 学年番号は数字か
21:        ($sec,$min,$hour,$day,$mon)=localtime; # 受信時刻の記録
22:        $mon++;                                # 1 増やすと月になる
23:        $in_data=join(",",@mdata).",$mon-$day $hour:$min:$sec"; # データと時刻
```

```
24:         last;                                   # 以降の行はスキップ
25:     }
26: }
27:
28: if($in_data eq ""){exit;}                       # 有効なデータがなければ終了
29: ############################ Step3 学生名簿ファイルの読込
30: chdir($workdir);                                # 作業 directory の変更
31:
32: open(L,"$std_list") || die "Cannot open student-list file\n"; # ファイルを開く
33: while(<L>){
34:     chop;                                       # 行末の改行コードの削除
35:     ($num,$gnum,$name)=split(/,/);              # 学年番号, 学生番号, 氏名の取得
36:     $gnum[$num]=$gnum;                          # 学年番号 -> 学生証番号の配列を定義
37:     $name[$num]=$name;                          # 学年番号 -> 氏名の配列を定義
38: }
39: close(L);                                       # ファイルを閉じる
40: ############################ Step4 学年番号と学生証番号の照合
41: if($in_gnum ne $gnum[$in_num]){exit;}           # 一致しなければ終了
42: ############################ Step5 一時ファイルへの追記保存
43: open(T,">> tmp/$tmp_file") || die "Cannot open tmp file\n"; # 追記モードで開く
44: print T "$in_num,$name[$in_num],$in_data\n";  # データ (学年番号, 氏名, 身長など) 書込
45: close(T);                                       # ファイルを閉じる
46: ############################ Step6 データの並べ替え
47: open(T,"tmp/$tmp_file") || die "Cannot open tmp file\n"; # 一時ファイルを開く
48: while(<T>){
49:     chop;                                       # 行末の改行コードの削除
50:     ($num)=split(/,/);                          # 学年番号の取得
51:     $data[$num]=$_;                             # 学年番号 -> データの配列を定義
52: }
53: close(T);                                       # ファイルを閉じる
54:
55: $data[0]=$data_item."\n\n";                      # タイトル行の設定
56: for $num (0..150){                              # 学年番号順に処理
57:     if($data[$num] ne ""){                      # 学年番号に対応するデータが空でなければ
58:        $final.="$data[$num]\n";                 # 変数$final にデータを順次格納
59:     }
60: }
61: ############################ Step7 データ・ファイルへの書き込み
62: open(F,"> $data_file") || die "Cannot open data file\n";  # ファイルを開く
63: print F "$final";                               # 変数$final の内容を書き込む
64: close(F);                                       # ファイルを閉じる
```

end of maildata_hf.pl

5. 2種類以上のデータ収集への拡張

実践医学統計の授業では, 既に紹介した身長・靴のサイズを始めいろいろなデータを収集している. 以下にその一覧を挙げる. 2行目は電子メールで送信するデータの書式の例で, 1行に収まらないものは途中を ... で省略してある. 3行目以降にはデータの説明と授業への利用目的などが書いてある.

- 身長・靴のサイズ
  `data:height,001,SD04002,164.5,23.0,end`
  身長の分布のヒストグラムを描いたり, 身長と靴のサイズの相関関係を調べる.
- 数学テスト
  `data:mathematics,001,SD04002,1,1,0,0,1,1,1,1,1,0,...,1,1,1,0,1,1,No,end`
  25題について各問ごとの正解・不正解を 1, 0 と数値化したデータを表集計の練習材料にしたり, 各問の正答率からどのような問題に誤答が多いかを分析する. さらに, 26項目のデータは「解答時間に余裕があったかどうか」を Yes,No で記述したものであり, これを利用して, 時間余裕派と時間不足派の2つのグループの間に学力差があるかを検定する.
- 漢字テスト
  `data:kanji,001,SD04002,1,1,1,1,1,1,0,1,1,1,1,1,1,...,0,1,1,1,1,0,0,0,end`
  数学テストと同様の観点から統計処理の練習をする.
- 脈拍
  `data:pulse,001,SD04002,35,35,39,34,35,34,end`
  脈拍を平常時や軽い運動, 激しい運動した直後など異なった状態で6回測定し, 関連2群の差の検定を行う.
- ジャンケン
  `data:janken,001,SD04002,0,0,0,-1,-1,1,0,-1,0,0,-1,1,1,1,1,1,1,0,-1,-1,0,end`
  ジャンケンを20回行い, 勝ち・負け・引き分けをそれぞれ 1, -1, 0 と数値化したデータを用いて, 大数の法則などの大標本の理論の理解に役立てる.

次が数学テストのデータ収集で得られるデータ・ファイルの例である.

データ・ファイル `mathtest04.txt`

```
1:  number,name,1,2,3,4,5,6,7,8,9,10,11,12,13,14,...,21,22,23,24,25,Y-N
2:
3:  001,秋葉　太郎,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,0,0,1,1,1,1,1,1,1,0,1,1,No
4:  002,東　京子,1,1,1,1,0,1,1,1,1,1,1,0,1,1,1,1,0,1,1,1,1,1,1,1,0,1,0,yes
5:  003,上野　公夫,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,0,1,0,No
6:  004,岡地　真知子,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,Yes
7:  005,御茶ノ水　博士,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,Yes
```

身長と靴のサイズ以外のデータの収集を行えるようにするには, 基本的には `maildata_hf.pl` の6-9行目の一時ファイル `$tmp_file`, データのタイトル行 `$data_item`, データ・ファイル `$data_file`, データ項目の個数 `$dnum` の値を書き換えて別のファイル名で保存して利用すればよい. しかし, この方法ではデータの種類ごとにメールアドレスを用意して, 受信メールが対応するスクリプ

ト・ファイルに渡るように設定する必要があるし, スクリプト・ファイルに変更を加えたいときはすべてのファイルを修正しなければならないので大変である.

そこで, maildata_hf.pl を複数のデータ収集に対応できるように改良する. 改良を加えたスクリプトはファイル名を maildata2.pl として, この節の末尾に掲載してある. 7 つの Step からなる全体の流れは同じだが, 次のような変更が加えられている.

- メール・データの先頭部分 (例 data:height, data:mathematics) からデータの種類を判別して (45 行目の変数$head), データの種類に応じて, データ項目の個数 (49 行目), 一時ファイル (78,82 行目), データ・ファイル (97 行目), データのタイトル行 (90 行目) が切り替わるようにした.
  これらの変数は Step1 でハッシュを用いてデータの種類ごとに設定してある.
- メール・データの先頭部分が間違っている場合は, 不正データとみなしてデータを破棄するように設定した. そのために, Step1 においてデータの各種類に 1 を対応させるハッシュ %flag を定義している. 不正データの破棄は, キーの存在しないハッシュの値は 0 になることを利用して, 46 行目で行っている.
- 年度を超えて対応できるように, 授業年度の西暦下 2 桁からなる変数$year を用意し, 学生名簿ファイルとデータ・ファイルに年度を付加した (4,5,9,16,23,29,36 行目).
- 数学テストのデータ・チェックでは, 第 26 項目が Yes または No になっているかを判定するようにした (52-55 行目). 大文字・小文字の区別はしないように, 送信された文字列を lc 関数で小文字に変換してから比較判定している.

スクリプトの改良に伴い, メール・サーバ側では次の変更作業を行う必要がある.

- /home/kyoin/tokei に maildata2.pl を置き, ファイル属性を読込可・書込可・実行可にする.
- メール・エイリアスの変更: ファイル/var/qmail/aliase/.qmail-tokei-en を次のように書き換える.

  | /home/kyoin/tokei/maildata2.pl

- /home/kyoin/tokei に mathtest04.txt, kanjitest04.txt などのデータ・ファイルを作成し, ファイル属性を読込可・書込可にする.
- /home/kyoin/tokei/tmp に tmp-math.txt, tmp-kanji.txt などの一時ファイルを作成し, ファイル属性を読込可・書込可にする.

スクリプト maildata2.pl

```
1:  #!/usr/bin/perl
2:  ############################ Step1 変数の設定
3:  $workdir="/home/kyoin/tokei";              # 作業 directory
4:  $year="04";                                # 年度 (西暦の下 2 桁)
5:  $std_list="std_list".$year.".txt";         # 学生名簿ファイル
6:
7:  $tmp_file{"height"}="tmp-hf.txt";          # 一時ファイル
```

```
 8:  $data_item{"height"}="number,name,height,feet,time"; # データのタイトル行
 9:  $data_file{"height"}="hf".$year.".txt";      # データ・ファイル
10:  $dnum{"height"}=2;                           # データ項目の個数
11:  $flag{"height"}=1;                           # データの種類が存在するかの判定用ハッシュ
12:
13:  $tmp_file{"mathematics"}="tmp-math.txt";
14:  $data_item{"mathematics"}="number,name,1,2,3,4,5,6,7,8,9,10,11,12,13,";
15:     $data_item{"mathematics"}.="14,15,16,17,18,19,20,21,22,23,24,25,Y-N";
16:  $data_file{"mathematics"}="mathtest".$year.".txt";
17:  $dnum{"mathematics"}=26;
18:  $flag{"mathematics"}=1;
19:
20:  $tmp_file{"kanji"}="tmp-kanji.txt";
21:  $data_item{"kanji"}="number,name,1,2,3,4,5,6,7,8,9,10,11,12,13,";
22:     $data_item{"kanji"}.="14,15,16,17,18,19,20,21,22,23,24,25";
23:  $data_file{"kanji"}="kanjitest".$year.".txt";
24:  $dnum{"kanji"}=25;
25:  $flag{"kanji"}=1;
26:
27:  $tmp_file{"pulse"}="tmp-pulse.txt";
28:  $data_item{"pulse"}="number,name,1,2,3,4,5,6";
29:  $data_file{"pulse"}="pulse".$year.".txt";
30:  $dnum{"pulse"}=6;
31:  $flag{"pulse"}=1;
32:
33:  $tmp_file{"janken"}="tmp-janken.txt";
34:  $data_item{"janken"}="number,name,1,2,3,4,5,6,7,8,9,10,11,12,13,";
35:     $data_item{"janken"}.="14,15,16,17,18,19,20";
36:  $data_file{"janken"}="janken".$year.".txt";
37:  $dnum{"janken"}=20;
38:  $flag{"janken"}=1;
39:  ############################ Step2 メール・データの整合性チェック
40:  while(<>){
41:      chop;                                    # 行末の改行コードの削除
42:      s/ //g;                                  # 不要なスペースの削除
43:      if(/^data/){                             # 行頭に data があるか
44:         ($header,$in_num,$in_gnum,@mdata)=split(/,/);  # 各変数へ代入
45:         ($heada,$head)=split(/:/,$header);    # $head に data:xxxx の xxxx の部分を代入
46:         if($flag{$head} ne 1){exit;}          # $head が登録された種類でなければ破棄
47:         $end=pop(@mdata);                     # @mdata の末尾を切り取り左辺に代入
48:         if($end ne "end"){exit;}              # 最後の項目が end か
49:         if(@mdata != $dnum{$head}){exit;}     # データ項目の個数が合っているか
50:         if(length($in_num) != 3){exit;}       # 学年番号は 3 桁か
51:         if(!($in_num=~ m/[0-9][0-9][0-9]/)){exit;} # 学年番号は数字か
52:         if($head eq "mathematics"){           # 数学テスト・データのチェック
53:             if(lc($mdata[$dnum{$head}-1]) ne "yes" &&
54:                lc($mdata[$dnum{$head}-1]) ne "no"){exit;}
55:         }
56:         ($sec,$min,$hour,$day,$mon)=localtime;# 受信時刻の記録
57:         $mon++;                               # 1 増やすと月になる
58:         $in_data=join(",",@mdata).",$mon-$day $hour:$min:$sec"; # データと時刻
59:         last;                                 # 以降の行はスキップ
60:      }
61:  }
62:
63:  if($in_data eq ""){exit;}                    # 有効なデータがなければ終了
```

```
64:  ############################ Step3 学生名簿ファイルの読込
65:  chdir($workdir);                              # 作業 directory の変更
66:
67:  open(L,"$std_list") || die "Cannot open student-list file\n";# ファイルを開く
68:  while(<L>){
69:      chop;                                     # 行末の改行コードの削除
70:      ($num,$gnum,$name)=split(/,/);            # 学年番号, 学生番号, 氏名の取得
71:      $gnum[$num]=$gnum;                        # 学年番号 -> 学生証番号の配列を定義
72:      $name[$num]=$name;                        # 学年番号 -> 氏名の配列を定義
73:  }
74:  close(L);                                     # ファイルを閉じる
75:  ############################ Step4 学年番号と学生証番号の照合
76:  if($in_gnum ne $gnum[$in_num]){exit;}         # 一致しなければ終了
77:  ############################ Step5 一時ファイルへの追記保存
78:  open(T,">> tmp/$tmp_file{$head}") || die "Cannot open tmp file\n";#追記モードで開く
79:  print T "$in_num,$name[$in_num],$in_data\n";# データ (学年番号, 氏名, 身長など) 書込
80:  close(T);                                     # ファイルを閉じる
81:  ############################ Step6 データの並べ替え
82:  open(T,"tmp/$tmp_file{$head}") || die "Cannot open tmp file\n"; # 一時ファイルを開く
83:  while(<T>){
84:      chop;                                     # 行末の改行コードの削除
85:      ($num)=split(/,/);                        # 学年番号の取得
86:      $data[$num]=$_;                           # 学年番号 -> データの配列を定義
87:  }
88:  close(T);                                     # ファイルを閉じる
89:
90:  $data{0}=$data_item{$head}."\n\n";            # タイトル行の設定
91:  for $num (0..150){                            # 学年番号順に処理
92:      if ($data[$num] ne ""){                   # 学年番号に対応するデータが空でなければ
93:         $final.="$data[$num]\n";               # 変数$final にデータを順次格納
94:      }
95:  }
96:  ############################ Step7 データ・ファイルへの書き込み
97:  open(F,"> $data_file{$head}") || die "Cannot open data file\n"; # ファイルを開く
98:  print F "$final";                             # 変数$final の内容を書き込む
99:  close(F);                                     # ファイルを閉じる
```

end of maildata2.pl

謝辞

本論文の一部は文部科学省学術フロンティア推進事業による.

参考文献および関連 URL

1) 日本の Linux 情報, http://www.linux.or.jp/
2) Plamo Linux, http://www.linet.gr.jp/~kojima/Plamo/
3) qmail, http://www.qmail.jp/qmail.html
4) Apache, http://httpd.apache.org/
5) 瀬戸 賢一 (2002) 日本語のレトリック (岩波ジュニア新書 418). 岩波書店, 東京
6) Wall L, Schwartz RL (1991) Programming perl. O'Reilley & Associates, California : 近藤 嘉雪 訳 (1993) Perl プログラミング. ソフトバンク, 東京

7) 結城 浩 (2003) Perl 言語プログラミングレッスン入門編. ソフトバンク, 東京
8) 藤田 郁, 三島 俊司 (1999) CGI & Perl ポケットリファレンス. 技術評論社, 東京
9) 河野 真治 (1994) 入門 Perl. アスキー出版局, 東京
10) qkc, http://hp.vector.co.jp/authors/VA000501/index.html