Perl プログラミング

大家 清治[1]

1 はじめに

テキストデータ (Text datas) などを単純に加工したり整形する処理には，どのような言語を使用すれば良いであろうか．このような処理のためのプログラミング (Programing) 言語を選択する場合に，「シェルスクリプト (*Shell script*) で記述するには荷が重すぎるし，C 言語で書くのは面倒すぎる．」と感じることがある．「気軽に容易に記述できるプログラミング言語は無いものであろうか．」というニーズ (Needs) に答えることができる最適な言語がある．それは Perl (*Practical Extraction and Report Language*：パールと読む) である．

Perl の解説書がまだ数少ないため，Perl プログラミングを修得しようとする場合に小さな学習障壁が従来まで生じていた．この学習障壁を取り払うべく，本解説では，Sample を多く掲載し，その解説を読み進んでいく過程で，Perl の文法体系が自然に理解できるように配慮した．ここでは，初級プログラミングの経験がある方を対象とし，入門レベル (Level) の細かい文法および制御構造の解説は他の適書に譲り，必要最低限度の解説にとどめた．

1.1 表記法

1. 太い枠内に実行操作と結果を，細い枠内に例題もしくは補足を示した．
2. 解説に対応する Index （番号）を例題枠内の各行の先頭の 4 カラム (Column) に付加した．各例題を実行する場合は，この部分を除いて入力されたい．
3. 既に解説済のスクリプト行およびそれに類似する行についての解説は省略した．
4. 半角の ¥ （エン）と \ （逆 slash）は同じ意味を持つ．実際の端末で使用できる方に置換して使用されたい．
5. この解説で用いた Perl の Version は 4.0.1.4 (Patch level 10) である．

[1]情報科学センター， ooie@isci.kyutech.ac.jp

2 Perl 概説

2.1 概要

2.1.1 Perl とは

Perl は 近年になって開発されたフリー・ソフトウェア (Free Software) である．Perl は，その使い良さから人気が高く，最もトレンディ (trandy) な言語と言われている． 開発者は Larry Wall 氏[2]である．

Perl は，通常なら sed や awk や sh を用いるような比較的に単純な処理を，もう少し軽く速く行ないたいが，Cで細かく記述するのは面倒と感じる場合に，最適な言語である．Perl は，C，awk，sed，sh 言語の最も良いところを結合する意図を持って開発された言語であるので，これらの言語に馴染みがある方は Perl 言語の習得を比較的に容易に感じることであろう．Perl の表現法は Cの表現法に似ている．

Perl は，大型汎用機のコマンド・プロシジャ（Command Procedure，例えば，ＲＥＸＸ) に類似したインタプリタ (Inter-preter) 型言語であり，データの処理を一行単位（複数行単位も可能．Binary datas の場合は File 単位）で行なう．Perl インタプリタは実行前に全ての文法をチェック (Check) する（同じインタプリタ型言語であっても，ＢＡＳＩＣでは実行直前の行についてのみ文法チェックが行なわれる）．

Perl で扱うデータ (Datas) の大きさには制限がない（ただし，使用しているマシン (Machine) の使用可能な Memory の大きさには制限される）．また，再帰の深さにも制限がない．Perl は，データフロー (Data flow) 追跡機構を持つので，Security holls を避けることができる．

Perl のデータ記述形式で記述したプログラムを *Perl Script* (ペル・スクリプト）と呼ぶ．スクリプトはテキスト (Text) 形式でファイル (File) として作成するか，あるいは，コマンドライン (Commnad line) から直接に指定入力を行なう．

Perl のバージョン・アップ (Version-up) や Patch level の修正は，現在も逐次行なわれており，機能が少しずつ拡張され続けている．

---

[2] <lwall@jpl-devvax.jpl.nasa.gov>　(注) Perl プログラミングに関する初歩的な質問事項について，開発者へ Direct Mail で問い合わせることは遠慮されたい．

2.1.2 Perl に向く処理

Perl は文字列の操作や検索処理に向いており，sed，awk が持つ機能をそのまま使用することができる．また，Perl は優れた Network 通信機能を持っており，Subroutine も使用できる．これらの点において，類似言語の awk より優れている．Perl は中規模程度の処理に向く．単純な処理であれば，*shell* でプログラムした方が簡単である．また，高度で複雑な処理であれば，C 言語 でプログラムした方が良い．Perl の全ての数値演算には倍精度浮動少数点型が用いられる（浮動少数点型演算を行なうと，大きな Over head をマシン (Machine) に強いることになる）．よって，Perl は複雑な算術演算やビット (Bit) 演算を行なう処理には向かない．このような処理は，Perl よりもＦＯＲＴＲＡＮでプログラムする方が良い．

2.1.3 変数の型とデータ

Perl では，変数と配列の型（文字型もしくは数値型）を宣言する必要はない．すなわち，変数と配列に格納するデータは，数値 (numeric) でも文字列 (string) でも構わない．それらの型は，文脈から自動的に類推される．それらが評価される時に，演算子によってそのデータは区分され，文字列値と数値に自動的に変換される．

Perl の変数と配列は原則的にグローバル (Grobal) であり，特に指定していない変数と配列の値は，Main routine と Subroutine との間で共通である．Subroutine の中の変数もしくは配列を独立させる場合は，local 関数を用いて局所変数（配列）の定義を行なう．こうすると，局所変数（配列）と Main routine の変数（配列）名が重複していても相互に独立した変数（配列）として使用できる．

2.1.4 Perl の特徴

以下に Perl の特徴をまとめる.

- インタプリタ型言語である.
- 特に文字列操作機能が豊富であり，パターンマッチング (Pattern maching) に正規表現 (Regular Expression) を使用できる.
- 実行開始時に簡単なコンパイル (Compile) をするらしく，実行速度は十分に速い．パターンマッチング処理ではC言語の標準関数よりも速いことがある.
- 文法や構造が C, awk, sed, sh に似ている．表記法は C および *shell script* に類似し，2行目以降はフリー・フォーマット (Free format) である.
- サブルーチン (Subroutines)，local 変数，ライブラリ (Library) を使用できるので，プログラムの記述が容易である．また，省略時の予約変数を利用して短い行数でスクリプトを記述できる.
- Cのライブラリの活用，および，ネットワーク (Network) 通信プログラミングが行なえる.
- バイナリ・データ (Binary datas) や dbm ファイルを扱うことができる.
- ほとんどの処理を Perl 自身で行なえるので，プロセス (Process) を増さずに処理できる.
- Security holls などへの Scurity 対策が整っているので，System prograing に適している.
- 使用機種に依存することが少ない（現時点では，UNIX と MS-DOS マシン，および，Mac で使用可能である）.

2.2 Perl スクリプトの表記方法

1. スクリプトを格納するファイル名は任意であり，拡張子（Suffix）は不要である.
2. 1行目先頭カラムには "#!" （シャープ・バン）に続けて Perl のインタプリタ本体のフルパス (Full path) を記述する．2行目からはフリースタイルで実際のプログラムを記述する.
3. 記述は半角文字で行なう（だだし，データとしての全角文字は使用できる）．大文字と小文字は区分される.
4. 各文（関数）の末には原則として ";" （セミコロン）が必要である.
5. 注釈（comment）を始めるカラムの先頭に "#" （シャープ）を指定する．この記号の右側の文字列は全て注釈となり実行に影響しない．注釈指定はその行においてのみ有効である.
6. 空白行を設けることができる．スクリプト内容を読み易くする場合に用いる.

2.3 Perl スクリプトの実行操作

Perl スクリプトファイルを作成して，実行する手順を以下に述べる．

2.3.1 スクリプトファイルの作成と保存

(1) テキストファイルを開く．．．

エディタ Emacs を使用してファイル Ex00 を開く例を以下に示す．

```
% emacs -nw Ex00
```

(2) 開いたファイルに以下のスクリプトを記述する．

Perl スクリプトのシンプルな例を以下に示す．これは，標準出力（画面）に Hello! と表示するものである．なお，例題枠内の各行の先頭の番号は，解説に対応させた番号であるので，これらの番号は入力しないこと．

例題 1: 基本スクリプト (basically script)(Ex00)

```
1  #!/usr/local/bin/perl
2  #  Sample script Ex00
3
4  print "Hello!\n";
```

1. 先頭2文字には，必ず，"#!" に続けて，Perl のインタプリタ本体のフルパスを記述する．（カラム間に空白カラムを設けないこと．）
2. 2行目以後はフリー・フォーマット．この例では，2行目に注釈行を設定した．
3. 良み易くするために空白行を設定した．
4. print 関数を使用して Hello! と標準出力（画面）に出力して改行する．なお，print 関数末には ";" を必ずつける．

(3) 内容を保存してファイルを閉じる．

1. 内容を以下の要領で保存（強制保存）する．

   最初に，Ctrl キー (Key) を押したままの状態で，英小文字の x を押す．

   次に，Ctrl キーを押したままの状態で，英小文字の s を押す．

2. このファイルを閉じる．

   最初に，Ctrl キーを押したままの状態で，英小文字の x を押す．

   次に，Ctrl キーを押したままの状態で，英小文字の c を押す．

2.3.2 ファイルへの実行および読取りの許可

Perl スクリプトを実行するためには，そのスクリプト・ファイルの許可モードの Read と Execute に許可を与えておく必要がある．UNIX 標準のコマンドの chmod を使用してファイルの許可モードを変更する．Current Directry のファイル **Ex00** の所有者 (user) による読み取りと実行を許可する操作例を以下に示す．

```
% chmod u+rx Ex00
%
```

2.3.3 実行操作

スクリプト・ファイル名を コマンドラインに入力して， Perl スクリプトを実行する[3]．
文法エラーがあれば，エラーメッセージ (Error messages) を標準出力へ出力し，コンパイルを中断する． 実行例を以下に示す．

```
% Ex00
Hello!
%
```

2.3.4 トラブル (Troubles) 対処

実行結果が前述のように表示されない場合の主な原因として，以下の 2 点が考えられる．

1) 入力内容が違っている（入力 Miss）．
   対処：細かい部分について例題とを再照合を行なう．誤りを修正して再度実行する．

2) Perl のインタプリタ本体 (`/usr/local/bin/perl`) が存在しない．
   対処 1：使用マシン上の Perl のインタプリタ本体の正しいフルパスを指定する．
   対処 2：「 Perl の導入」について，利用マシンの管理者に相談する．

[3] （参考）デバッグと実行操作．
X端末から利用している場合は，スクリプトの修正用 Window と 実行用 Window の 2 つを同時に表示すれば，デバッグ処理がより容易になる． tty 端末で利用する場合に同時使用画面数を複数にすることができる (screen コマンド)．ただし，表画面に表示できる画面数は一つであり，裏画面を順次切替えて表示を行なう．この方法の詳細については，情報科学センターへ問い合わせされたい．

3 基本的な処理

3.1 変数と配列 (Variable and Array)

変数と配列を使用して，簡単な英文を標準出力（画面）に表示する例を以下に示す．

(1) 実行結果

スクリプト・ファイル名の入力後，以下の文[4]が標準出力（画面）に表示される．

```
% Ex01a
宮沢 喜一 さん は 56 歳です.
宮沢 りえ さん は 21 歳です.
宮沢 健治 さん は 16 歳です.
%
```

(2) スクリプトおよび解説 （新出：変数，配列， print ）

変数名の頭は $ ，配列名の頭は @ で始める．詳細については補足 2(配列と変数) を参照されたい．

例題 2: メッセージ出力スクリプト (Message outputting) （変数と配列） (Ex01a)

```
 1  #!/usr/local/bin/perl
 2  # Sample script Ex01a
 3
 4  $family = "宮沢";
 5  @FIRST = ("喜一","りえ","健治");
 6  @AGE = (56,21,16);
 7
 8  for ( $j=0; $j <= 2; $j++ ) {
 9    print "$family @FIRST[$j] さん は @AGE[$j] 歳です. \n";
10  }
```

1. 第 1 行目には，必ずこの文を指定する．
2. 注釈文．
3. 読み易くするために空白行を設定した．
4. 変数 $family へ文字列 "宮沢" を代入する．
5. 配列 @FIRST へ要素 "喜一"， "りえ"， "健治" を代入する．
6. 配列 @AGE へ要素 56,21,16 を代入する．
8. 変数 $j の値に 0 を設定し以下の処理を行なう．その後，変数値に 1 を 加えて同様に処理を行なう．変数値が 2 になるまでこれを繰り返す．
9. print 関数では，配列 @FIRST の添字 $j の要素，変数 $family の値， 配列 @ARGE の添字 $j の要素を出力し改行する．

[4]例題で使用した氏名は架空のものであり，実在する人物の氏名とは無関係である．

3.2 文字変数での文字列の扱い

同じ内容の行を6行表示するスクリプト例を以下に示す．（出力行数6に注目されたい．）

例題 3: メッセージ出力スクリプト（文字列結合）(Ex01b)

```
 1  #!/usr/local/bin/perl
 2  # Ex01b    (William Shakespeare の作品「Hamlet」第3幕 第1場から)
 3
 4  $A = "or not to be:";
 5  $B = "To be, ". $A ." that is the question:". "\n";
 6  $C = $B x 3;
 7
 8  print "To be, $A that is the question:\n";
 9  print "To be, ", $A ," that is the question:", "\n";
10  print $B;           # （訳1）行なうべきか，行なわざるべきか，それが問題だ．
11  print $C;           # （訳2）生きるべきか，死ぬべきか，それが問題だ．
```

6. 代入文の右式 $B x 3 は，変数 $B の内容の文字列 (改行コード (Code) を含む) を3回分つないだものと等価である．これについての詳細は，補足1（print 関数）および補足5（演算子． i. 繰り返し演算子，j. 文字列結合演算子）を参照されたい．

補足 1: print 関数

```
1) print の右側に，出力する変数もしくは文字列を ，(カンマ) でつないで記述する．
  a) 文字列はクォートで囲む．
      "   (ダブルクォート) に囲まれた部分での変数はその要素に置換される．
      '   (シングルクォート) に囲まれた部分での変数はそのままの文字列と解釈される．
  b) 文字列の出力後に改行したい場合は，その文字列の末尾に改行コードを付加する．
      \n    改行コード                \f    改頁
      \t    TAB コード                \r    復帰
  c) 例：次の3つの print 関数は同じ内容を出力する．
      print A$,B$,"出力文字 ","$C Message\n";
      print A$.B$."出力文字 "."$C Message\n";   # "."(ピリオド) は文字列の結合演算子．
      print "A$B$ 出力文字 $C Message\n";
2) print の右式を省略した場合は，省略時予約変数 $_ の内容が出力される．例： print;
3) FILEHANDLE に対して出力する場合は次のように記述する．
        (FILEHANDLE STDOUT) に出力する例：    print STDOUT "Message\n";
4) 書式付きの出力を行ないたい場合は，次のいずれかの方法を用いる．
  a) printf 関数．(これについては，補足9を参照されたい．)
  b) format 関数で定義しておいて， write 関数を使用する．
        (format を使用する場合，漢字などの2バイトコードの2バイト目の書式が途中で
         無くなった場合には，そのコード以降の出力に文字化けを生じることがある．)
5) その他．(ASCII コード (8進数値) の出力方法)
     例：  print "\101";   # "A" を表示する．(参考："A" のコードの16進数値は 41)
           print "\07";    # ベルを鳴らすコード．
           print "\04";    # Ctrl+D のコードを出力する．
```

補足 2: 変数と配列 (Variable and Array)

1) 変数と配列の型（文字型もしくは数値型）を宣言する必要はない.
それらは，文脈から自動的に類推される．必要に応じて，文字列値と数値が自動的に変換される.

変数と配列に格納するデータは，数値 (numeric) でも文字列 (string) でも構わない.
その変数が評価される時に，演算子によってそのデータが区分けされる.
例えば，等価を比較する場合の，文字列の比較演算子は "eq" であり，
数字の比較演算子は "==" である.
演算子についての詳細については補足 5 (演算子) を参照されたい.

2) 変数名の頭文字には "$"（ダラー）をつける.
3) 配列名の頭文字には，一般に，"@"（アットマーク）をつける．（例外あり．）

a) 配列全部や配列の複数リスト (Lists) を取り出す場合は，配列名の頭文字には "@" をつける.
b) 配列の単数リストを取り出す場合は，配列名の頭文字に "$" をつけることもできる.
配列リスト @A[0] の参照を $A[0] で代替できる.
変数 $A と $A[0] は別のものとして区別される.
c) 連想配列の場合で，全データを参照する場合は，配列名の頭文字に "%" をつける.
（連想配列の詳細については，補足１９（連想配列と環境変数）を参照されたい．）

4) 変数と配列名には，英文字と数字，および，いくつかの記号 "-"（マイナス），
"_"（アンダーバー）などを使用できる.
これらの名称では，大文字と小文字は区分される.

5) 配列の添字は 0 から始まる．例えば，配列 @A の第一要素はリスト @A[0] に格納される.

6) 配列の複数指定方法は以下のとおりである.
@A[0,2] は @A[0],@A[2] を意味し，@A[0..2] は @A[0],@A[1],@A[2]
を意味する.
@A は その全要素を返す.

7) 配列の全要素を捨てて，空にする場合は，次のように行なう． 例： @A = ();

8) 配列のリストの数の参照例を次に示す. 例： print $#A;
$#A は配列 @A に使用している最後のリストの番号を持つ.
リストは 0 から始まるので，リスト数はこの番号に 1 を加えた値になる.

4 情報検索処理の基本

Perl は，データ行の中から指定した文字列を検索し，その検索結果に応じて処理を区分する処理に優れている．Perl では，パターン・マッチングに正規表現（Regular Expression）を用いることができる．

4.1 データ入出力の基本操作

4.1.1 基本例

ファイル・データから文字列を検索する例を以下に示す．

(1) 実行結果

入力ファイルの内容を cat で出力[5]し，その出力をパイプで繋ないで，スクリプト・ファイル Ex02a の標準入力へ転送する．Ex02a での処理結果は標準出力 (画面) へ表示[6]される．

```
% cat /etc/passwd | Ex02a
Root information = root:eJpeaxbtT7YFg:0:1:Operator:/:/bin/csh
%
```

(2) スクリプトおよび解説 （新出：while, chop, split, if ）

入力データ・ファイル[7]の第 1 項目が root であるデータ行を検索し，その行内容を標準出力に表示する例を以下に示す．

例題 4: 検索スクリプト (Pattern maching) （基本） (Ex02a)

```
 1  #!/usr/local/bin/perl
 2  # Ex02a
 3
 4  while (<STDIN>) {
 5     chop;                        # 左の部分を各々，以下のように記述しても良い．
 6     split(/:/);                  #  @LINE = split(/:/,$_);
 7     if ( @_[0] eq "root" ) {     #  if ( @LINE[0] eq "root" ) {
 8        print "Root information = $_\n";
 9     }
10  }
```

[5]ネットワークが NIS （ Network Information Service ）で管理されている場合は，実際のシステム・ファイルから検索を行なうために，入力部の操作を cat /etc/passwd から ypcat passwd に置換して実行すると良い．

[6]標準出力（画面）に表示される実行結果は，実行マシン毎に異なる．

[7]システム・ファイル（ /etc/passwd ）の各項目は ":" （コロン）で区切られた自由長であり，その構成は以下のとおりである．

Login 名 : 暗号化パスワード : ユーザー番号 : グループ番号 : フルネーム : 起動シェルのフルパス

1. 最初の行には，必ずこの文を指定する．
2. 注釈文．
4. 標準入力（STDIN）からデータを一行読み込み，そのデータ行を 省略時の出力予約変数 $_ に格納し，入力データが無くなるまで繰り返す．
5. 入力データの末尾コード（多くの場合，末尾には改行コードがある）を削除する．
6. 省略時の入力予約変数 $_ の内容から，":" で区切った要素を，省略時の出力予約配列 @_ のリスト @_[0] から 添字の昇順に格納していく．
7. 配列 @_ の１番目のリスト @_[0] の要素値が root であれば，以下の行が実行される．
8. メッセージと変数 $_ の内容を標準出力へ出力後，改行する．

補足 3: split 関数

```
split(' ');  複数の空白を区切り記号とする.
split(/ /);  一つの空白を区切り記号とする．入力データに複数の空白が含まれている場合は,
               空白数分の null 要素が生成される.
split(/\s/); 空白を区切り記号とする.
split(/\b/); 単語の終りを判定してそれを区切りを記号とする．まれに，空白も単語の一部
               として認識されることもある.
```

なお，split(/ /); を使用する場合は，必要に応じて，前処理として tr/ / /s; を使用する方が良い．

tr 関数についての詳細は，補足１７（パターンマッチング (2)）を参照されたい．

補足 4: 正規表現 (Regular Expression)

正規表現は，パターンマッチングに用いられる記法である．その基本的な規則を以下に示す．

^	行の先頭にマッチする．	*	直前の文字列の０個以上の繰り返しにマッチする．
$	行の末尾にマッチする．	+	直前の文字列の１個以上の繰り返しにマッチする．
.	任意の１文字にマッチする．	?	直前の文字列が０あるいは１個にマッチする．

[abc]	文字 "a" または "b" または "c" にマッチする．
[a-z]	文字 "a" から "z" までのいずれかにマッチする．
[^abc]	文字 "a" と "b" と "c" 以外にマッチする．
[^a-z]	文字 "a" から "z" までのいずれか以外にマッチする．
y|n	文字 "y" または "n" のいずれかにマッチする．
(yes|no)	文字列 "yes" または "no" のいずれかにマッチする．
\	直後のメタキャラクタ１個を普通の文字として扱う．"\." はピリオド文字１個の意味．
m	直後の文字で囲まれた範囲をパターンの範囲とする．m 指定省略時の文字は / である．

\w	[_a-zA-Z] と同じ処理を行なう．	\b	単語の境界にマッチする．
\W	\w 以外の文字にマッチする．	\B	\b 以外の文字にマッチする．
\d	[0-9] と同じ処理を行なう．	\s	一つの空白文字にマッチする．[ ] と同じ処理．
\D	\d 以外の文字にマッチする．	\S	\s 以外の文字にマッチする．

補足 5: 演算子 (Operator)

a) 比較演算子（文字列）
eq は文字列の等価を判定する．なお，文字列の比較に == 演算子は使用できない．
cmp は文字列を比較して，左側が大であれば 1，等価であれば 0，小であれば -1 を返す．
その他に，ne , le , ge , lt, gt がある．

b) 比較演算子（数値）
== は数値の等価を判定する．
!= は数値が等価でないことを判定する．
<=> は数値を比較して，左値が大であれば 1，等価であれば 0，小であれば -1 を返す．
その他に，<= , >= , < , > がある．　なお，比較演算子として <> は使用できない．

c) 算術演算子 （C言語のオペレータ (Operator) と同じ．詳細はCの Manuals を参照されたい．）
+　-　*　/　%　+=　-=　*=
/=　%=　++　--
** べき乗　**= べき乗して代入

d) ビット演算子 （C言語のオペレータと同じ．詳細はCの Manuals を参照されたい．）
&　|　^　~　>>　<<　&=　|=　^=
~=　>>=　<<=

e) 論理演算子 （条件式で使用される演算子）
|| （ダブルパイプ）は論理和を意味する．
&& （ダブルアンパサンド）は論理積を意味する．
! （バン）は論理値を否定したものを返す．
補足 7 （条件式の評価），補足 1 3 （論理演算式と論理値）も参照されたい．

f) 代入演算子
= 右式の値を左式の変数などに代入する．

g) 検索対象変数設定演算子
=~ （イクォール・チルダ）は，パターンの検索置換を省略時予約変数 $_ 以外の変数に対して行なう場合に用いる．
この演算子の使用方法については，補足 8 (パターン・マッチング (1)) を参照されたい．
(対象文字列) =~ (検索・置換式);
!~ （バン・チルダ）は，=~ 演算子の戻り値を否定する場合に用いる．

h) 範囲指定演算子
.. 配列のリストの指定では，左の値から右の値まで増分 1 で増加する値をリストとする．

i) 繰り返し演算子
x 左側の文字列（変数）を右側の回数分繰り返した文字列を返す．
x= 文字列を繰り返して代入する．

j) 文字列結合演算子
. 左側と右側の文字列（文字変数内容）を結合する．
.= 文字列を結合して代入する．

k) ファイル・テスト (File test) 演算子
-e ファイルが存在することを判定する．存在する場合は真となり 1 を返す．
-r ファイルを実効 uid で読めることを判定する．
-w ファイルを実効 uid で書けることを判定する．
-T ファイルがテキストファイルであることを判定する．
ファイルテスト演算子として，その他に，　-x -o -R -W -X -O
-z -s -f -d -l -p -S -b -c -u -g -k -c -B -M -A -C　がある．

4.1.2 簡略化例

(1) スクリプトおよび解説 (新出：print-if)

スクリプト Ex02a に省略時の変数を利用して簡潔に表現したスクリプト Ex02b を以下に示す．このスクリプトの実行結果は Ex02a の実行結果と同じになる．

例題 5: 検索スクリプト (簡略化) (Ex02b)

```
1  #!/usr/local/bin/perl
2  # Ex02b
3
4  while (<>) {
5    split(/:/);
6    print "Root infomation = $_" if ( @_[0] eq "root" );
7  }
```

4. 標準入力 (STDIN) の指定は省略できる．
   データ末コードを chop を用いて除かずに処理できる．
5. 省略時の出力予約配列 @_ = split(/:/, 省略時の入力予約変数 $_);
6. 実行式を左式に，条件式を右式に記述できる． 実行式が一文の場合はこの方法が良い．

4.1.3 コマンドラインからの直接実行

スクリプト Ex02b と同じ処理を awk などのようにコマンドラインから直接に実行することができる[8]．その例を以下に示す．このスクリプトの実行結果は，Ex02a の実行結果から "Root information = " のメッセージ部分を除いたものと同じになる．

```
% cat /etc/passwd | perl -ne 'split(/:/); print if (@_[0] eq "root");'
```

4.1.4 正規表現の利用例

正規表現を使用して，csh を使用している利用者の id を表示する処理例を以下に示す．

例題 6: 検索スクリプト (正規表現) (Ex02c)

```
1  #!/usr/local/bin/perl
2  # Ex02c
3
4  while (<>) {
5    print "$1\n"  if (m@^([^:]+):.*/csh$@);
6  }
```

5. パターンの引用符は @ である． [^:]+ は ： 以外の文字の 1 個以上の繰り返しにマッチする． ( ) で括ったパターンにマッチした文字は 変数 $1 に格納される．それに続いて，さらに任意の文字列が続き，その末尾が /csh であればマッチする．詳しくは，補足 4 (正規表現) を参照されたい．

[8] コマンドラインでは，行末に \ を入力すれば，複数行の入力ができる．

4.2 データをファイルから入力

ファイルからデータを読み込み，指定パターンを含むデータ行を抽出する方法を考える．第1に，ファイルからデータ行を1行づつ読み出しながらパターン・マッチングを行なう方法がある．第2に，ファイルの全データ行を全て配列に蓄え，その後，その配列の全要素についてパターン・マッチングを行なう方法がある．この2つの方法による処理時間の比較実験を行なった例を以下に示す．

(1) 実行結果

あるデータ・ファイル（10万レコード，総計 7.32 MByte）を入力データ・ファイルとして，実行した結果[9]を以下に示す．第1の方法を STEP 1 で表示し，第2の方法を STEP 2 で表示した．実行結果の枠内の左側の数値はユーザー時間，右側の数値はシステム時間である．

```
% Ex03
STEP 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Begin =   0.016666666666666666435          0.050000000000000002776
Find! =ooie:LK5ZNeOumlKQ6:1005:12100:Kiyoharu OOIE:/home/ooie:/bin/csh
End   =  53.916666666666664298191          1.416666666666666740682
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
T(s)  =  53.899999999999998578915          1.366666666666666696273
STEP 2 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Begin =  53.916666666666664298191          1.416666666666666740682
Find! =ooie:LK5ZNeOumlKQ6:1005:12100:Kiyoharu OOIE:/home/ooie:/bin/csh
End   =  94.900000000000005684342          9.849999999999999644729
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
T(s)  =  40.983333333333341386151          8.433333333333333570181
%
```

この実行結果では，第2の方法での処理時間の方が短い．第2の方法では，全データを配列に格納してからマッチングを行なう．よって，データ量が多い場合はシステム (System) のメモリ (Memory) を圧迫し，マシン全体に大きな負荷を与える．

[9] パターンにマッチする行数が1行の場合の times 関数による処理時間の測定値．
マシン: Sun Sparc Station 2.　単位: 秒．

(2) スクリプトおよび解説

(新出：do, sub, open, close, times, printf, /pattern/, grep)

例題 7: 検索スクリプト（応用）(Ex03)

```
 1  #!/usr/local/bin/perl
 2  # Ex03
 3
 4  $infile = "/etc/passwd";
 5  $pattern = "ooie";
 6
 7  do start("STEP 1");
 8    while(<IN>) {
 9       print "Find! =$_" if ( /($pattern):/ );
10    }
11  do end();
12
13  do start("STEP 2");
14     @PASSWD = <IN>;
15     print "Find! =@_" if ( @_ = grep(/$pattern:/, @PASSWD) );
16  do end();
17
18  sub start {
19    ($user1,$system1,$dummy,$dummy) = times;
20    printf ("%s%s\nBegin = %25.21f\t%25.21f\n",
21                           $_[0],' -' x 27,$user1,$system1);
22    open(IN, "< $infile") || die "Can't open $infile: $!\n";
23  }
24
25  sub end {
26    close(IN);
27    ($user2,$system2,$dummy,$dummy) = times;
28    printf ("End   = %25.21f\t%25.21f\n%s\n",$user2,$system2,' -' x 30);
29    printf ("T(s)  = %25.21f\t%25.21f\n",
30                  ($user2 - $user1),($system2 - $system1));
31  }
```

4. 変数 $infile に 入力データ・ファイル名をフルパスで指定する.
5. 変数 $pattern に 文字検索を行なうための 文字列 "ooie" を指定する.
7. サブルーチン start の処理を行なう. 引数として "STEP 1" を渡す.
8. (FILEHANDLE IN) からのデータ行が無くなるまで繰り返す.
9. データ行を省略時予約変数 $_ へ読み込み, 以下の処理を行なう.
   条件式が真の場合に 実行関数を実行し, 変数 $_ の内容を表示する. 変数 $pattern のデータ末に ":" を追加した文字列 (この例では "ooie:") が, パターン・マッチでの省略時入力予約変数 $_ の 内容に含まれている場合に, 条件式は真となり 1 を返す.
   変数名を正しく示すために, 変数 $pattern を "()" (カッコ) で囲む.
11. サブルーチン end の処理を行なう. 引数は無い.

13. サブルーチン start の処理を行なう．引数として "STEP 2" を渡す．

14. (FILEHANDLE IN) からの入力データを配列 @PASSWD に一行ずつ蓄積する． データ行が無くなるまで，この配列の添字０をもつリスト @PASSWD[0] から昇順の添字を持つリストへ順次格納する．

15. grep は，配列 @PASSWD の全要素 の内容から，変数 $pattern のデータ末に ":" を追加した文字列（この場合， "ooie:" ）を含むデータ行を抽出して，配列 @_ に順次格納する． 配列 @_ の要素数が 0 でない場合に条件式は真となり 1 を返す． 条件式が真の場合に配列 @_ の全要素を出力する．

18. サブルーチン start を設定する．

19. このスクリプトの実行時から現時点までの実行時間を time 関数で測定する．
(ユーザ使用時間, システム使用時間, 不使用データ，不使用データ) = 時間測定関数

20. 書式付き出力関数である．
printf （ 自由長文字型書式，自由長文字型書式，改行，実数型書式小数点以下２２桁， TAB，実数型書式小数点以下２２桁，改行 ）;
なお， ' -' x 27 は， ' -' を２７回分並べた文字列になる．

22. 変数 $infile で指定したファイルを (FILEHANDLE IN) に定義し，こんファイルからデータ読み込む設定を行なう．このファイルへのアクセスができない場合に限って，エラー情報予約変数内容などを出力して，強制中断する．

23. サブルーチン start の終りを設定する．

25. サブルーチン end を設定する．

26. (FILEHANDLE IN) に設定したファイルを閉じる．

31. サブルーチン end の終りを設定する．

補足 6: サブルーチン (Subroutine)

1) サブルーチンは以下のように記述する．サブルーチン名は任意である．
```
sub SUBROUTINE_NAME { 処理関数 };
```
2) サブルーチン・コールには do 関数，または， & 形式を用いる．
```
例： do SUBROUTINE_NAME(LIST);
     &SUBROUTINE_NAME(LIST);
```
3) do 関数では，サブルーチンの実行後，サブルーチンで実行された最後の expression の値を返す．
4) 括弧の指定．
& 形式では，引数を渡さないのであれば，括弧を使う必要はない．
do 関数では， do EXPR との混乱を防ぐため，括弧が必要である．
5) サブルーチンに渡した引数は全て 配列 @_ に格納される．
6) サブルーチンから引数を返す例を次に示す．
```
&SUBROUTINE_NAME(LIST);        # コール時に LIST 部分に格納変数名を指定する．
sub SUBROUTINE_NAME {          # サブルーチンを設定する．
   return LIST;                # リターン関数に値を設定する．
}
```
再帰関数や局所変数を用いた場合に引数を利用する．
変数および配列は local 関数で指定しない限り，サブルーチンとメインルーチン間で共通．
7) return 関数の指定について．
return を明示すると若干遅くなる．引数無しの場合は， return 関数を省略できる．

補足 7: 条件式の評価 (Evaluation of conditional expression)

条件式の評価は左から行なわれ，その式全体の論理値が確定した所で打ち切られる．
（判定式１） || （判定式２） || （判定式３）
つまり，　（判定式１）が 真であれば，（判定式２）の評価は行なわれない．
逆に，　（判定式１）が 偽であれば，（判定式２）以降の評価に移る．
&& の場合も同様である．

このことから，以下の open 関数の右式の制御方法が理解できよう．

```
open(IN, "< $infile") || die "Can't open $infile: $!\n";
```

補足 8: パターン・マッチング (Pattern maching) （１）

/PATTERN/;
省略時の予約変数 $_ の文字列の中の PATTERN を検索して，真偽を返す．
検索対象変数を指定する場合は，"=~" 演算子の左に変数を指定する．
"=~" 演算子については，補足５（演算子，g．検索対象変数指定演算子）も参照されたい．

検索対象変数として $A を指定する例：

```
$A =~ /PATTERN/;
```

補足 9: printf 関数の変換書式 (Print Format)

以下の変換書式を使用できる．

%c	ASCII 文字．
%d	１０進数．
%e	[-]d.dddddddE[+-]dd
%g	e 変換と f 変換のどちらか短い方，無意味なゼロを抑止する．
%s	文字列．
%o	符号無し８進数．
%x	符号無し１６進数．
%%	% 文字を一つ印字．

なお，% の後に 以下のパラメータ (Parameters) を付加できる．

-	左揃え．
数字	指定した数字の桁を確保．
. 数字	小数点の右側桁数，または，文字列の最大桁．

補足 10: open 関数と ファイルハンドル (File handle) （1）

open 関数の記述は以下のように行なう.

```
open(FILEHANDLE,EXPR);
```

例： /etc/passwd を入力用ファイルとし, FILEHANDLE 名として IN を設定する.

```
open(IN,"< /etc/passwd");
```

1) EXPR で指定したファイルを開いて, FILEDANDLE 名に結び付ける.

2) EXPR で指定する文字列の最初の記号は以下の意味をもつ.

<	入力用ファイルを設定する. （指定しない場合は入力用と認識される. ）
>	出力用ファイルを設定する.
>>	追加用ファイルを設定する.
+>, +<	Read と Write 両方可能なファイルを設定する.
|	コマンドと解釈し, 出力がそのコマンドにパイプされる.
-	STDIN （標準入力） をオープンする.
>-	STDOUT （標準出力） をオープンする.

3) EXPR で指定する文字列の最後の記号は以下の意味をもつ.

|	コマンドと解釈し, そのコマンドからの入力がパイプされる.

4) EXPR に複数のパイプ (Pipe) を使用することもできる.

a) パイプをひとつ使用する例を以下に示す.
変数 $home で指定した ディレクトリ (Directry) のファイルから,
検索パターンに照合するファイルの属性を ls -al で表示する.

```
open(IN,"ls -al $home |");
while(<IN>) {
    next if ( substr($_,0,1) ne "-" );
    if ( /$pattern/ ) { print; }
}
```

b) パイプを複数使用する例を以下に示す.
スクリプトの第1引数で指定したディレクトリのファイルから, 第2引数で指定した
文字列を含むファイルの属性を ls -al で表示する.

```
open(IN," ls -al $ARGV[0] | grep -e '\^-' | grep -e '$ARGV[1]' | ");
    while(<IN>) { print; }
```

補足 11: close 関数

close 関数は以下のように記述する.

```
close(FILEHANDLE);
```

open で設定した FILEHANDLE 名に対応させて記述する.

5 応用処理

5.1 外部プログラムとのデータ渡し と スクリプトの引数

現在のプロセス情報を UNIX 標準コマンド の ps によって得て，スクリプトの引数で指定した文字列を含むプロセスをそのアルファベット昇順に sort して出力する例を以下に示す．

(1) 実行結果

スクリプト Ex04 の引数に文字列 "ooie" を指定して実行した例を以下に示す．

```
% Ex04 ooie
ooie       5245  -sh (csh)
ooie       5904  -sh (csh)
ooie       6388  /usr/local/bin/perl Ex04 ooie
ooie       5253  emacs -nw title.v2
ooie       6391  grep -e ooie
ooie       5219  oclock -bg black -bd white -hour white -jewel white -minute\
 white -geometry 80x80-20+15
ooie       6392  ps -auxww
ooie       6390  sh -c ps -auxww | grep -e 'ooie'
ooie       6389  sort -t +2
ooie       5212  twm
ooie       5220  xbiff -geometry 75x75-22+120
%
```

（2）スクリプトおよび解説 （新出： @ARGV, $#ARGV, substr ）

例題 8: プロセス (Process) 情報の整形表示スクリプト (Ex04)

```
 1  #!/usr/local/bin/perl
 2  # Ex04
 3  #       Usage:  Ex04  Key-word
 4
 5  $in_grep = "";
 6  if ( $#ARGV != -1 ) { $in_grep = "| grep -e '$ARGV[0]'"; }
 7
 8  open(OUT,"| sort -t +2 ");
 9  open(IN," ps -auxww $in_grep | ");
10
11  if ( $#ARGV eq -1 ) { <IN>; }
12  while(<IN>) {
13     split(' ');
14     printf (OUT "%-8s%7d  %s",@_[0..1],substr($_,56));
15  }
16
17  close(IN);
18  close(OUT);
```

5. 変数 $in_grep に 初期値 "" (null) を代入しておく.
6. 条件式は, 予約変数 $#ARGV の値が -1 以外 (引数の数が0) の場合に真となる.
   条件式が真の場合に, 変数 $in_grep に文字列を格納する. ここで, "!=" は, 数値が等価でないことを判定する比較演算子である. ' (シングルクォート) で囲まれた文字列に含まれる変数 $ARGV[0] は, その要素に変換されずに, そのままの文字列として解釈される.
8. (FILEHANDLE OUT) を出力として設定する. ここからの出力を UNIX 標準コマンドの sort -t +2 の処理に渡して, その出力を標準出力へ出力する. sort -t +2 は, 空白で区切られた3番目の文字列を昇順にソート (Sort) する.
9. (FILEHANDLE IN) に対して, UNIX 標準コマンドの ps の出力を渡す設定をする.
   " (ダブルクォート) で囲まれた文字列に含まれる変数 $in_grep は, その要素に変換されて解釈される.
11. 不要ヘッダーを読み飛ばす処理を設定する. 引数無しの場合には, UNIX の標準コマンドの ps -auxww を実行することによってヘッダー (Header) が出力される.
12. データを読み込む.
13. 省略時の入力予約変数 $_ の内容から複数の空白 (スペース) で区切った要素を, 省略時の出力予約配列 @_ のリスト @_[0] から 添字の昇順に格納していく.
14. printf 関数の出力を (FILEHANDLE OUT) に対して送る. %-8s は左詰め8桁の文字書式である. 配列 @_[0..1] は 配列 @_ の添字0から1までを表す. substr($_,56)[10] は 変数 $_ の 57 文字目以後の文字列を出力する.

このスクリプトは, 現在の処理のプロセスとその番号を加工表示するものである. これは, 実行中の自分のプロセスを強制終了したい場合に役立つ. この場合, その処理のプロセス番号を調べて次の入力を行なえば良い.

```
% kill -KILL プロセス番号
%
```

補足 12: スクリプトの引数

a) スクリプトに引数を渡す場合は,
   スクリプト・ファイル名の右側に引数を空白で区切って並べて, スクリプトを実行する.
   例: % Ex99 coffee cake sugar

b) 引数の格納は以下のように行なわれる.

   空白を区切りとして区分され, 予約配列 @ARGV のリスト (添字0のリストから) に順次格納される. つまり, 引数の最初の文字列は, @ARGV[0] に格納される.

c) 引数の数は以下のように参照できる.
   予約変数 $#ARGV に格納される. つまり, この変数 $#ARGV 値は,
   引数が無い場合は -1, 引数の数が 1 の場合は 0 , 2 の場合は 1 である.

[10] substr の詳細については, 項 7.2(エッセンス表示) を参照されたい.

5.2 システム・コール (System call) と 予約変数

UNIX の標準コマンドの mv では，ファイル名にワイルドカード（あいまいな指定）を用いることはできない．しかし，スクリプトで処理を行なえば，一連の規則的な名称がついているファイル群を，別の規則的な名称がついたファイル群に，1回のコマンド入力で移動させることが可能となる．ここでは， mv example.* example.*.old のようなファイル名の変更処理の意図を実現するスクリプト[11]を紹介する．

（1）実行結果

移動元ファイル群として，カレント・ディレクトリ (Current directry) の先頭文字が (,7) で始まる全てのファイルを指定し，移動先ファイル群として，先頭のカンマを取った新しいファイル（ 7* ）を指定した例を以下に示す．

```
% Ex05 ',7*' '7*'
mv  ,7  to  7
mv  ,70 to  70
mv  ,71 to  71
mv  ,72 to  72
mv  ,73 to  73
mv  ,74 to  74
mv  ,75 to  75
mv  ,76 to  76
mv  ,77 to  77
mv  ,78 to  78
mv  ,79 to  79
%
```

（2）スクリプトおよび解説（新出：$', $`, tr///, =~, if-elsif-else, system, rename）

ワイルドカードを用いた rename 処理スクリプトを以下に示す．

例題 9: ワイルドカードを用いた RENAME スクリプト (Ex05)

```
1  #!/usr/local/bin/perl
2  # Ex05
3  #       Usage:  Ex05  '*.FOR' '*.f'
4
```

（次ページに続く）

[11]このスクリプトは，メール・リーダー (Mail reader) MH を使用してメールを整理していた際に，保存すべきメール群を rmm コマンドで大量に削除してしまったミスを回復するために作成した．rmm コマンドの実行によって削除されたメールは "," を先頭に付加したファイルとして暫定的に残される．
例えば， mv ,2* 2* の処理を実行したい時に役立つ．

```
 5  if (( $ARGV[0] =~ tr/*/*/) <=> 1 || ($ARGV[1] =~ tr/*/*/) <=> 1 ) {
 6       print "Options not mached!\n";
 7       do help();
 8       exit;
 9  }
10  $ARGV[0] =~ /\*/;     $xf = $`;        $xb = $';
11  $ARGV[1] =~ /\*/;     $yf = $`;        $yb = $';
12
13  foreach $x (<$ARGV[0]>) {
14     $y = $x;
15
16     if     ( $xf eq "" && $yf ne "" ) { $y = $yf.$y; }
17     elsif  ( $xf eq "" && $yf eq "" ) { ; }
18     else   { $y =~ s/$xf/$yf/o; }
19
20     if     ( $xb eq "" && $yb ne "" ) { $y = $y.$yb; }
21     elsif  ( $xb eq "" && $yb eq "" ) { ; }
22     else   { $y =~ s/$xb$/$yb/o; }
23
24     print "mv  $x\tto  $y\n";
25     # system("mv -i $x $y");        # Process are produced eachtime
26     # rename($x, $y);               # No process is produced
27  }
28  exit;
29
30  # -- Subroutine --
31
32  sub help {
33     print <<EOF;
34  $0 Usages:
35   【名前】
36           $0 - ワイルドカードを用いた rename 処理スクリプト
37   【形式】
38           $0  '入力ファイル名'  '出力ファイル名'
39   【オプション】
40           入出力ファイル名を各々，クォートで囲んで指定する．
41           "*" を入出力ファイル名の中に各々一つだけ含めて指定する．
42   【例】
43           $0 '*.FOR' '*.f'          ex:      abc.FOR --> abc.f
44           $0 ',2*' '2*'             ex:      ,234    --> 234
45           $0 'a*.d' 'e*.h'          ex:      abc.d   --> ebc.h
46  EOF
47  }
```

5. 【不当オプション (Option) の判定】
2つの引数の中の "*" (アスタリスク) の数が，各々1でない場合の処理を以下のように行なう．
if (条件式1 論理和 条件式2) { 実行関数 ; }
論理和を取った条件式が真の場合に 実行関数を処理する．
条件式1では，変数 $ARGV[0] の中に含まれる "*" の数を出力する．
条件式2では，変数 $ARGV[1] の中に含まれる "*" の数を出力する．
$ARGV[0,1] は スクリプトの引数を格納している予約配列のリストである． 比較演算子 "!=" は 数値を比較し，等しくない場合に真となり 1 を返す．

7. サブルーチン help を実行する．引数渡しは無い．

8. このスクリプトの実行を終了する．

10. 【ワイルドカードを用いない文字列の抽出】
変数 $ARGV[0] の要素について，最初の "*" の位置を検索． アスタリスクを普通の文字として使用していることを認識するために， 頭に制御コード "\" (逆スラッシュ，または，半角の "¥") を付加する． 制御コードをつけない場合は，この "*" は特別な意味をもつ．
最後の検索でマッチした文字列の前の文字列を 変数 $xf へ代入する．
最後の検索でマッチした文字列の後に続く文字列を 変数 $xb へ代入する．

13. 【移動元ファイル群のそれぞれのファイル名の抽出と処理】
変数 $ARGV[0] の要素の文字列を含むファイル名を入力として， 該当したファイル名を一つずつ 変数 $x に渡して， foreach 関数の範囲内の処理をデータが無くなるまで繰り返す．

14. 変数 $x の内容を変数 $y へ代入する．

16. 【変換後のファイル名の作成 (1)】
ファイル名のワイルドカード指定部分の左側の文字列の指定について， 変換前は null で 変換後が null でない場合に， null でない部分を有効にする．

17. 変換前および変換後が null である場合には加工しない．

18. それ以外の場合は，変数 $y の内容に対して，変換前の文字列を変換指定の文字列に一回だけ置き換える処理を行ない，その置換結果を変数 $y に格納する．

20. 【変換後のファイル名の作成 (2)】
ファイル名のワイルドカード指定部分の右側の文字列の指定について，前と同様な処理を行なう．

24. 変換前と変換後のファイル名を標準出力に出力する．

25. 【実際の処理】
意図に反した実行防止のため，コメントにしている．実際の処理を行なう場合は， "mv" と "rename" のいずれか一方を選択して， 該当行上の "#" を外して，注釈文を解除する．

system (/bin/sh -c に渡すコマンド);
このシステム・コール関数を実行する度に mv のプロセスを毎回起動するので非効率である． しかし， mv -i の -i オプションによって， 変換先ファイル名が既に存在する場合には， そのファイルへの over-write の是非をインタラクティブ (Interactive: 会話的) に尋ねてくるので， スクリプトのバグ (Bugs) などがある場合に誤って処理されることを防ぐ意味でより確実である．

26. rename(OLDNAME,NEWNAME);
ファイルの名前を変更する．この関数を実行することでプロセスが増えることはないので， 処理は効率的である． しかし， 変換しようとするファイル名が既に存在する場合であっても， そのファイルへの over-write の是非をインタラクティブに確認せずに， 強制的に処理をおこなうので注意が必要である．

28. このスクリプトの終了を定義する．

32. サブルーチン help を定義する.

33. この関数以後, 最初に "EOF" の文字列が出現するまでの範囲行を print 関数のデータとする定義を行なう. この範囲では, 変数や配列も指定でき, 改行コード ("\n") を設定する必要はない.

34. このスクリプトのファイル名が予約変数 $0 に格納されている.

46. 前述の print 関数のデータ範囲の終了を宣言する文字列は "EOF" である. なお, "EOF:" という文字列では判別されないので注意されたい.

補足 13: 論理演算式と論理値 (Boolean Value)

1) 以下の式の左辺と右辺は, ド・モルガンの法則により, 等価となる.

```
!$A && !$B = !($A || $B)
!$A || !$B = !($A && $B)
```

なお, !$A は $A の否定である.

2) 式が, 数値データとして0を持つか, あるいは, 文字列データとして "" (null) を値に持つ場合に偽となる. それ以外の場合は真となり 1 を返す.

補足 14: 予約変数 (Predefined Names)

主な予約変数を以下に示す. 予約変数はこの他にもある.

```
$_      入力およびパターン検索における省略時変数.
@_      サブルーチンで受けた引数値を格納する配列.
@ARGV   スクリプトに与えられた引数値を格納する配列.
$ARGV   <> から読み込んでいる時のカレントファイル名.
%ENV    環境変数が設定してある連想配列.
$`      (ダラー・逆シングルクォート)
        最後のパターン・マッチングでマッチした文字列の前の文字列が格納されている.
$&      最後のパターン・マッチングでマッチした文字列が格納されている.
$'      (ダラー・シングルクォート)
        最後のパターン・マッチングでマッチした文字列の後に続く文字列が格納されている.
$1       "( )" で括った1番目のパターンにマッチした文字列が格納されている.
$2       "( )" で括った2番目のパターンにマッチした文字列が格納されている.
        $3 の場合も同様である.
$$      実行中 (自身) の Perl スクリプトのプロセス番号.
$>      実行中 (自身) の Perl スクリプトのプロセスの実効 uid.
$0      実行中 (自身) の Perl スクリプトのファイル名が格納されている.
$?      パイプのクローズ, system 関数などで帰ったステータス (Status) 値.
$!      errno の現在値 または system error 文字列.
$@      最後に行なった eval の syntax error message.
```

補足 15: foreach 関数

```
foreach 変数名 ( 配列名 ) { 実行関数 ; }
```

foreach 関数は, 配列の要素を一つ取り出して変数に格納し, 配列要素が無くなるまで実行関数の処理を繰り返す.

補足 16: ファイルハンドル（2）

FILEHANDLE 名には，普通の変数と区別するために，大文字だけを用いる方が良い．

<IN>　FILEHANDLE 名として IN を指定した例である．

<STDIN>　標準入力からの入力．予約 FILEHANDLE として， STDIN, STDOUT, STDERR がある．

<>　標準入力，または，コマンドラインにリストアップした全ファイルから入力を行なう．

カレント・ディレクトリのファイル *.c に該当する全ファイルの permission を変更する例を次に示す：

```
chmod 0755, <*.c>;
```

<変数>　変数を指定できる．

1) 変数に設定している文字列が FILEHANDLE 名となる場合の例を以下に示す．

例： FILEHANDLE 名として IN を指定する例：

```
<$foo>
```

```
open(IN,"< test");
$foo = "IN";
while(<$foo>) { print; }
```

2) 変数に設定している文字列が入力ファイル名となる場合の例を以下に示す．

強制的にファイル名置換を行なう場合は次のように {} を入れる：

```
<${foo}>
```

例： 変数に指定した文字列に該当する全ファイル名を印刷する．

```
$foo = "*.c";
while(<${foo}>) { print "$_\n"; }
```

補足 17: パターン・マッチング（2）（置換）

置換関数として s 関数と tr 関数 ならびに y 関数がある．y 関数は tr 関数と同じ機能を持つ．

1) s/ 検索パターン / 置換パターン /goie;

検索パターンを持つ文字列を検索し，最初にマッチした文字列のみ（ただし，オプション指定により全文字列についての検索と置換ができる）を置換パターンへの置換を行なう．

[オプション説明]（一部）

g　見つかった全ての検索パターンを置換する．

o　最初の検索パターンだけを置換する．

2) tr/ 検索パターン / 置換パターン /cds;

検索パターンを持つ文字列を全て検索し，置換パターンへの置換を行なう．

[オプション説明]

c　指定した検索パターン以外の文字列を検索パターンとする．

s　変換変換されて同じ文字になった文字列を 1 文字に縮める．

d　検索パターンの文字について，置換パターンに無い文字を全て削除する．

例： tr/　/ /s;　2つの半角空白を1つの半角空白に変換する．

tr/a-z0-9/ /c;　指定したパターン（小文字 a から z までと数字）以外の文字を半角空白へ置換．

3) 検索パターン文字の最後が "$" である場合は，その行の末尾から検索を行なう．

補足 18: システム・コール (System call)

system(LIST);　最初に fork を行ない，親プロセスは子プロセスが完了するのを待つ．

exec(LIST);　親プロセスは子プロセスが完了するのを待たない．他の点は system 関数と同じ．

5.3 連想配列 (Associative Array)

連想配列は Perl の優れた機能の一つである．連想配列では文字列をキーとして要素を格納できる．keys 関数で連想配列のキーを取り出すことができる．連想配列の特定単数データを扱う場合は連想配列名を "$" で始めるが，全データを扱う場合は連想配列名を "%" で始める．

（1）実行結果

以下のテキスト・ファイル text1 （日本語を含まない）を入力データ[12]として，英単語の使用頻度（アルファベット昇順，および，使用頻度降順）一覧表を出力するスクリプトを以下に示す．

```
% cat text1
Pat-a-cake, pat-a-cake, baker's man,          # ぺたぺたこねてよ おかしやさん
Bake me a cake as fast as you can;            # おかしをやいてよ おおいそぎ
Pat it and prick it, and mark it with B,      # たたいて つついて Bのじ おして
Put it in the oven for baby and me.           # オーブンにいれてよ おねがいだ.
%
% cat text1 | Ex06
 -- 英単語 昇順表示 --
a                       is(are)       3
and                     is(are)       3
as                      is(are)       2
b                       is(are)       1
baby                    is(are)       1
bake                    is(are)       1
baker's                 is(are)       1
cake                    is(are)       3
can                     is(are)       1
fast                    is(are)       1
for                     is(are)       1
          (中，省略)

 -- 使用頻度 降順表示 --
it                      is(are)       4
a                       is(are)       3
and                     is(are)       3
cake                    is(are)       3
pat                     is(are)       3
as                      is(are)       2
me                      is(are)       2
b                       is(are)       1
baby                    is(are)       1
bake                    is(are)       1
baker's                 is(are)       1
can                     is(are)       1
          (以下，省略)

%
```

[12]入力データの出典：Mother Goose の伝承童謡「Pat-a-cake （ぺたぺたこねてよ おかしやさん）」．

(2) スクリプトおよび解説 （新出：keys, %WORDS, $a++, value）

例題 10: 英単語使用頻度一覧表出力スクリプト (Frequency in words use)(Ex06)

```
 1  #!/usr/local/bin/perl
 2  #  Ex06
 3  #  英単語の使用頻度を表示するスクリプト（連想配列を使用）
 4
 5  while(<>) {
 6    chop;
 7    tr/A-Z/a-z/;
 8    tr/a-z0-9\'/ /cs;
 9    split(' ');
10    foreach (@_) {
11        $WORDS{$_}++;
12    }
13  }
14
15  print " -- 英単語 昇順表示 --\n";
16  foreach $key (sort(keys %WORDS)) {
17          printf "%-20s  is(are)  %5d\n",$key,$WORDS{$key};
18  }
19
20  print " -- 使用頻度 降順表示 --\n";
21  open(OUT,"| sort +2n -r ");
22     foreach $key (keys %WORDS) {
23          printf (OUT "%-20s  is(are)  %5d\n",$key,$WORDS{$key});
24     }
25  close(OUT);
```

5. 標準入力からデータを入力し，省略時予約変数 $_ に格納する．

7. 省略時の入力予約変数 $_ の内容に対して，全ての大文字を小文字に置換する．

8. 同様に，英小文字と数字と ' （シングルクォート）以外の文字を空白に置換する．日本語（2バイト文字）を出力対象に含める場合は，この行を注釈にすると良い．

9. 省略時の入力予約入力変数 $_ の内容を一つ以上の空白で区切り，省略時の出力予約配列 @_ に格納する．

10. foreach 関数の入力配列は @_ である．出力変数は省略時の出力予約変数 $_ である．

11. 連想配列 %WORDS のキーを変数 $_ とし，その要素に 1 を加える．

16. keys 関数で，連想配列 %WORDS の全ての文字列キー (Keys) をランダム (Random) な順序で出力する．このキーをソートした結果を foreach 関数で変数 $key に順次格納し，foreach 関数の範囲内の処理をデータが無くなるまで繰り返す．

17. 変数 $key を２０桁左詰めで，連想配列 %WORDS の指定キーの要素値を５桁で出力する．

21. (FILEHANDLE OUT) からの出力を，第３番目の項目について数値として降順にソートして，標準出力へ出力する．

22. 連想配列 %WORDS の全ての文字列キーを foreach 関数によって得る．(FILEHANDLE OUT) へ連想配列 %WORDS の要素値と，変数 $key のキーを出力する．

25. (FILEHANDLE OUT) で設定したファイルを閉じる．

このスクリプトの出力結果をファイルへ出力する場合は，スクリプトに若干の修正を加える必要がある．この修正を加えたスクリプトの修正例（部分）を以下に示す．

```
15  $outfile = "outlist";                          # 出力先ファイル名を設定する.
16
17  open(OUT,"> $outfile");                        # 出力用ファイルを設定する.
18  print OUT " -- 英単語 昇順表示 --\n";
19  foreach $key (sort(keys %WORDS)) {
20          printf OUT "%-20s  is(are)  %5d\n",$key,$WORDS{$key};
21  }
22  print OUT " -- 使用頻度 降順表示 --\n";
23  close(OUT);
24
25  open(OUT,"| sort +2n -r >> $outfile");        # 追加出力用ファイルを設定する.
26     foreach $key (keys %WORDS) {
27          printf (OUT "%-20s  is(are)  %5d\n",$key,$WORDS{$key});
28     }
29  close(OUT);                                    # Ex06b
```

補足 19: 連想配列と環境変数 (Associative Array and Environmental Variable)

1) 連想配列名として，特定単数データを扱う場合は "$" を先頭につけ，データ全体を扱う場合には "%" を先頭につける．
2) 連想配列名には，通常配列や変数と区別するために，大文字を用いる方が良い．
3) 連想配列のキーを keys 関数で，要素を values 関数で取り出すことができる．
   値はランダムに取り出される．
4) 連想配列のキーと要素の組を each 関数で取り出すことができる．組はランダムに取り出される．

5) 特定の key と value の組を削除する方法．
   連想配列 %WORDS のキー "cake" とその値の組を削除する例を以下に示す．

```
delete $WORDS{'cake'};
```

6) 連想配列の全データを削除する例を以下に示す．

```
undef %WORDS;
```

7) 環境変数を参照する例を以下に示す．
   予約連想配列 %ENV を用いる．

```
print $ENV{'PATH'};          # 環境変数 PATH の値を標準出力へ表示する例.
```

8) 環境変数を設定する例を以下に示す．
   ここでの設定は，Perl スクリプトの実行中のみ有効である．

```
$ENV{'EDITOR'} = `ng`;       # 環境変数 EDITOR への設定例.
system("printenv");          # 全ての環境変数値を表示する.
```

5.4 サブルーチンでの引数渡し

5.4.1 加算関数 (Addtional Function)

関数を用いて 4 + 9 の演算を行なう例を以下に示す.

(1) 実行結果

```
% Ex07
13
%
```

(2) スクリプトおよび解説 (新出：return)

サブルーチンでの処理結果をメインルーチンに渡す場合は, return にリストを引数として指定する. return 関数を含むサブルーチンを関数と呼ぶこともある.

ユーザー関数 add を定義して, 加算演算を行なうスクリプトを以下に示す.

例題 11: 加算関数 (Additional Function)(Ex07)

```
1  #!/usr/local/bin/perl
2  #  Ex07
3
4  print &add(4,9),"\n";
5  sub add { return @_[0]+@_[1]; }
```

4. 引数として 4 と 9 を関数 add に渡し, 関数の戻り値を標準出力へ出力し, 改行する.

5. サブルーチンで受けとった引数値は 予約配列 @_ へ順次格納される. 2 つの引数の和をメインルーチンに返す.

5.4.2 再帰関数 (Recursive Function)

関数を用いて 10*9*8*7*6*5*4*3*2(*1) の演算を行なう例を以下に示す.

(1) 実行結果

```
% Ex08
3628800
%
```

(2) スクリプトおよび解説 （新出：local）

例題 12: 再帰関数 (Recursive Function)(Ex08)

```
1  #!/usr/local/bin/perl
2  #  Ex08
3
4  print &factorial(10)."\n";
5  sub factorial {
6     local($n) = $_[0];
7     if ($n < 2 ) { return 1; }
8     return $n * &factorial($n-1);
9  }
```

4. 関数 factorial に引数 10 を渡し，関数の戻り値を標準出力に出力し改行する．

5. 関数 factorial を定義する．

6. 予約配列のリスト $_[0] の要素を local 関数で設定した局所変数 $n に代入する．

7. 局所変数 $n の値が 2 よりも小さくなれば，戻り値 1 でメインルーチンへ戻る．

8. 再帰演算を行なった結果の値を return 関数の引数でメインルーチンへ返す．

補足 20: local 関数

特に指定していない変数と配列の値は，メインルーチンとサブルーチンとの間で共通である．
サブルーチンの中の変数もしくは配列を独立させる場合は，local 関数を用いて局所変数（配列）の定義を行なう．こうすると，局所変数（配列）とメインルーチンの変数（配列）名が重複していても相互に独立した変数（配列）として使用できる．

例：

```
@ABC = ("A","B","C","D","E","F","G");
do test();
print "@ABC\n";

sub test {
  local(@ABC) = @ABC;
  shift(@ABC);              # 配列の最初の要素を取り去り，要素全体を左に一つずらす．
  print "@ABC\n";
}                           # ABC
```

この例の実行結果は次のとおりである．

```
B C D E F G
A B C D E F G
```

5.5 データベース (Data base)

Perl には，UNIX に標準で付加しているデータベース・ライブラリの dbm を使うためのインターフェイスが組み込まれている．dbm は基本的なデータベース機能を実現したもので，キーと値を関連させてファイルに保存し，キーによってその値を高速に検索するものである．

5.5.1 データベースの作成 (Make Database)

/etc/passwd ファイル・データから，login ID を連想配列のキーとして，これとユーザーの氏名との関連を作り，この関連をデータベース /tmp/db.dir，/tmp/db.pag に格納する例を以下に示す．

(1) 実行結果

実行後，2つのファイルが作成される．これらのファイルの属性を ls -alg コマンドで表示した例と合わせて，以下に示す．

```
% Ex09a
% ls -alg /tmp/db.*
-rw-r--r--  1 user     kyutech      4096 Aug  5 21:42 /tmp/db.dir
-rw-r--r--  1 user     kyutech      4096 Aug  5 21:42 /tmp/db.pag
%
```

(2) スクリプト （この例題スクリプトの解説は省略する．） （新出：dbmopen，dbmclose ）

例題 13: データベース作成スクリプト (Make Database)(Ex09a)

```
#!/usr/local/bin/perl
#  Make DB
#  Ex09a

$dbfile = "/tmp/db";
$infile = "/etc/passwd";

dbmopen(DB,$dbfile,0666) || die "Can't open $dbfile: $!\n";
open(IN,$infile)         || die "Can't open $infile: $!\n";

while(<IN>) {
        ($login,$passwd,$uid,$gid,$name) = split(/:/);
        $DB{$login} = $name;
}
dbmclose(DB);
```

5.5.2 データベースのアクセス (Access Database)

データベースを開いて連想配列の要素をアクセスすることで簡単に検索が行なえる.

(1) 実行結果[13]

```
% Ex09b
root      --  Operator
takefu    --  Masasuke Takefu
jin       --  Hitoshi Nakayama
ooie      --  Kiyoharu OOIE
anzai     --  hiroyuki anzai
yamanoue  --  Yamanoue takashi
 (以下, 省略)
%
```

(2) スクリプトおよび解説 (新出: each)

前述のスクリプトで作成したデータベースを検索するスクリプトを以下に示す.

例題 14: データベース・アクセス・スクリプト (Access Database)(Ex09b)

```
 1  #!/usr/local/bin/perl
 2  #  Use DB
 3  #  Ex09b
 4
 5  $dbfile = "/tmp/db";
 6
 7  die "Can't open $dbfile: $!\n" if ( !-e "$dbfile.dir" || !-e "$dbfile.pag" );
 8
 9  dbmopen(DB,$dbfile,0666);
10  while (($login,$name) = each %DB) {
11          printf ("%-8s  --  %s\n",$login,$name);
12  }
13  dbmclose(DB);
```

7. 条件式が真の場合に, メッセージを出力してスクリプトの実行を終了する.
   /tmp/db.dir と /tmp/db.pag のいずれかのファイルが存在しない場合に条件式は真となる[14].

9. dbmopen 関数では指定するファイルの入出力用途の区別を指定できない. 指定したファイルが存在しなければ, そのファイルを新規に作成する. 例では, 新規作成ファイルの permission を ugo+rw に設定した.

10. 連想配列 %DB のキーと要素の組を each 関数によって 変数 $login, $name へ一組ずつ格納する. 連想配列 %DB のデータが無くなるまで while 関数の範囲内の処理を繰り返す.

[13]使用しているマシンによって, 出力内容は異なる.

[14]解釈にド・モルガンの法則を適用した. 詳しくは, 補足 13 (論理演算式と論理値) を参照されたい.

5.6 ファイル管理 (File Manegement)

5.6.1 オーナー変更 (Change owners)

(1) 実行結果

/tmp/t* のファイルのオーナーとグループ (Group) を，/etc/passwd に登録されているユーザー "ooie" の uid と gid へ変更する例を以下に示す．ここで，uid と gid を入力する必要はない．なお，root 管理者以外の者が他人のファイルのオーナーを変更することは実際には行なえない．

```
% Ex10
Change owner to ? (User-ID): ooie
Object File-names ? (ex: /tmp/*.test) : /tmp/*
chown 1005.12100
Object = /tmp/apple /tmp/orgenge /tmp/banana /tmp/pineapple /tmp/tomato
%
```

(2) スクリプト （この例題スクリプトの解説は省略する．） （新出：chown）

例題 15: ファイルのオーナーを変更するスクリプト (Change files's owner)(Ex10)

```
#!/usr/local/bin/perl
#  Ex10
#  Change files owner and group to foo

$passwd = "/etc/passwd";

print "Change owner to ? (User-ID): ";
  $user = <STDIN>;
  chop($user);
print "Object File-names ? (ex: /tmp/*.test) : ";
  $pattern = <STDIN>;
  chop($pattern);

open(PASS,"$passwd") || die "Can't open $passwd: $!\n";
while (<PASS>) {
   ($login,$pass,$uid,$gid) = split(/:/);
    $uid{$login} = $uid;
    $gid{$login} = $gid;
}
close(PASS);

@ARY = <${pattern}>;                                    # get filenames
if ($uid{$user} eq '') { die "$user not in $passwd $!\n"; }
else {
   # chown $uid{$user}, $gid{$user}, @ARY;              # real process
   print "chown $uid{$user}.$gid{$user}\nObject = @ARY\n";  # test case
}
```

5.6.2 複数オーナーへの変更

特定の uid をオーナーとするファイルを検索し，このファイルのオーナーを対応マップ (Map) で指定した uid に変更する処理[15]を，複数オーナー分まとめて行なう例を以下に示す．

(1) 実行準備

あらかじめ，変換対応マップを作成しておく．対応マップファイルを /tmp/UMAP に作成した場合の例を以下に示す．

```
% cat /tmp/UMAP
1012 4003 from inoue to ohnishi
1005 5015 from ooie  to takefu
1002 6016 from jin   to tsujita
%
```

このマップファイルの第 1 行目は，オーナーが 1012 である該当ファイルのオーナーを 4003 に変更することを意味する．この行の空白で区切った第 3 項目以降は備考であり参照されない．第 2，3 行についても同様である．

(2) 実行結果

参照するトップ・ディレクトリを /tmp にした例を以下に示す．このディレクトリの下に存在する全てのファイルがオーナー変更の対象となる．なお，このスクリプトではファイルの gid の変更は行なわれない．

```
% find /tmp -ls | Ex11
chown 1002.12100 to 6016.12100  /tmp/test1.ohnishi
chown 1002.12100 to 6016.12100  /tmp/test2.ohnishi
chown 1005.12100 to 5015.12100  /tmp/test1.ooie
chown 1005.12100 to 5015.12100  /tmp/test2.ooie
chown 1005.12100 to 5015.12100  /tmp/ooie
chown 1005.12100 to 5015.12100  /tmp/passwd
chown 1005.12100 to 5015.12100  /tmp/db.pag
chown 1005.12100 to 5015.12100  /tmp/db.dir
chown 1005.12100 to 5015.12100  /tmp/UMAP
chown 1012.12100 to 4003.12100  /tmp/test1.inoue
chown 1012.12100 to 4003.12100  /tmp/test2.inoue
%
```

[15]特定ディレクトリ下の全てのファイル群を特定単数のオーナーへ強制的に変更する場合は，UNIX の標準コマンド（ chown -R owner.group ）を用いる方がより簡単である．

(3) スクリプトおよび解説 （新出：stat, next-if, defined）

例題 16: 特定ファイル群を複数オーナーへ変更するスクリプト (Change files's owner)(Ex11)

```
 1 #!/usr/local/bin/perl
 2 # Ex11
 3 #  Usages: find Users-Top-Directry -ls | Ex11
 4
 5 $umap = "/tmp/UMAP";
 6
 7 open(UMAP,"< $umap") || die "Can't open map-file $umap: $!\n";
 8 while(<UMAP>) { split; $new_uid{$_[0]} = $_[1]; }
 9 close(UMAP);
10
11 while(<>) {
12    chop;
13    split(' ');
14    ($uid,$gid) = (stat($_[10]))[4,5];          # ファイルのステータスを返す.
15    next if !( defined($uid) && defined($new_uid{$uid}));
16    # chown $new_uid{$uid}, $gid, $_[10];                    # Real process
17    print "chown $uid.$gid to $new_uid{$uid}.$gid \t$_[10]\n"; # Test
18 }
```

8. 入力データを空白で区切り，第1項目の文字列をキーとする連想配列 $new_uid に，第2項目の文字列を格納する．

15. define 関数は，その要素の値が null でないこと（real value を持つこと）を判定し，真の場合は 1 を返す．この行の処理は，2つの要素について値の存在の判定を行ない，いづれかの要素が null であれば[13] while 関数のループの先頭に戻る．

16. 実際の処理を行なう．意図に反した実行を避けるため，注釈にしている．

17. 処理状況を標準出力へ表示する．

なお，この例題16（オーナー変更）を使用して，オーナー変更の処理を実際に行なう場合は，16行目の注釈 "#" を除いて実行されたい．例題15についても同様である．

6 その他のスクリプト例

使用頻度が高いと思われる処理を行なうスクリプトを以下に紹介する．なお，この項以降の例題スクリプトについては，詳細な解説を省略する．

6.1 対話型メニュー (Interactive Menu)

対話型入力を行なう場合の処理例を以下に示す．パターンマッチングに正規表現（補足4参照）を使用している．

```
#!/usr/local/bin/perl
#  Ex12a

print &answer(),"\n";

sub answer {
   print "OK(y/n/Quit/Sample/Memo/?)> "; chop ($answer=<STDIN>);
   if ($answer !~ /^[ \t]*[nNqQsSmM?hH]/) { return("0"); }
   if ($answer =~ /^[ \t]*[qQ]/) {
      print "中断しますか(y/n)? "; chop ($_=<stdin>);
      if (/^[ \t]*[yY]/) { exit; }
         else { return("2"); }
   }
   if ($answer =~ /^[ \t]*[mM]/)  { do print_howto();   return("2");}
   if ($answer =~ /^[ \t]*[sS]/)  { do print_example(); return("2");}
   if ($answer =~ /^[ \t]*[hH?]/) { do print_help();    return("2");}
   else { return("1"); }       # The case of [nN]
}
```

6.2 スクリプトの引数の参照

以下は，プリンタ・フィルタ (Printer filter) に使用したスクリプトの一部である．これを利用して，プリンタの lpd デーモン (Daemon) からの引数を得ることができる．パターンマッチングに正規表現（補足4参照）を使用している．

```
#!/usr/local/bin/perl
#  Ex12b        Usage:  Ex12b -w80 -l66 -n ooie -h aoi /var/adm/lpd-acct

   if ($ARGV[0] =~ /^[ \t]*[hH?]/ ) { do show_help(); exit; }
   while ($_ = $ARGV[0], /^-/) {
        $_ = shift;
        if    (/^-w/)   { $width = $';    next; }       # column number
        elsif (/^-l/)   { $length= $';    next; }       # line number
        elsif (/-n$/)   { $user  = shift; next; }       # user ID
        elsif (/-h$/)   { $host  = shift; next; }       # host name
   }
   $acctfile = shift;                                   # acct file name
```

6.3 安全なファイル入出力の方法

ファイル入出力を安全に行なう処理例（部分）を以下に紹介する．

(1) データ入力のためのファイルのオープン (Open) 処理例を以下に示す．

以下の２つのいずれかの方法で行なう．

```
open(IN, "< $infile")          || die "Can't open $infile: $!\n";     # Ex12c
if ( open(IN,"$infile") != 1 )  { die "Can't open $infile: $!\n"; }
```

(2) 追加用にテンポラリ・ファイル (Temporary file) を作成し，処理後，削除する例を以下に示す．

```
$tmp  = "/tmp/test.tmp$$";                  # Ex12d
                    # 予約変数 $$ には，このスクリプトのプロセス番号が格納されている．
unlink $tmp;                                # 同じファイルがあれば削除しておく．
system("touch $tmp");                       # null ファイルを作成する．
chmod 0660, $tmp;                           # ファイルの permissoin を設定する．
open(OUT,">> $tmp ");                       # データ追加用ファイルとして開く．
                                            #   （この行間に処理を記述する．）
close(OUT);                                 # ファイルを閉じる．
unlink $tmp;                                # ファイルを削除する．
```

(3) ファイルの存在状態を調べて，その存在と書き込み許可を評価する例を以下に示す．

ファイルが存在しないか，または，書き込み不可能であれば[13]，ファイルを追加用に新規作成する．そうでなければ，読み込みと書き込みの両方のモードでファイルを開く．

なお，ファイル・テスト演算子だけでファイルの状態を調べることができるので stat 関数を使用する必要はないが，参考として例に含めて示した．

```
$file = "/tmp/test";                        # Ex12e
($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,
        $atime,$mtime,$ctime,$blksize,$blocks) = stat($file);
                                            # stat は ファイルの状態を調べて返す．
if (!(-e $file && -w $file) ) {             # -e, -w は ファイル・テスト演算子．
        system("touch $file");
        chmod 0660, $file;                  # ファイルの Permission を ug+rw と設定．
        print "Made $file newly\n";
        open(INOUT,">> $file ");            # >>  は ファイルを追加用として設定．
}
else { open(INOUT, "+>> $file "); }         # +>> は read と write の両方を許可する．
```

6.4 ラベル (Label の使用例)

ラベルを使用したスクリプトの例を以下に示す.

```
#!/usr/local/bin/perl
# Ex12f

$pattern1 = "sync";      $pattern2 = "uucp";

goto STEP3 if (@ARGV[0] eq "e" );        # 第1引数が "e" の場合はラベル STEP3 へ飛ぶ.

STEP1:  for ($j=0 ; $j <= 2; $j++ ) {
STEP2:      while (<>) {
               last STEP1 if /${pattern1}/;
                       # 標準入力データが変数 $pattern1 の文字列に
                       # パターンマッチした場合に, STEP1 のループの最後へ飛ぶ.

               next if /${pattern2}/;
                       # 標準入力データが変数 $pattern2 の文字列に
                       # パターンマッチした場合に, ループの先頭 ( while ) へ飛ぶ.
               print;
            }
         }
STEP3:
  print "Ok! $_\n";
```

6.5 コマンドライン (Command lines) からの Perl 使用例

find で探したファイルを効率的に削除する方法について.

カレント・ディレクトリの *.tmp に該当するファイル群を削除する例を以下に示す.

```
% find . -name '*.tmp' -print | perl -ne 'chop; unlink;'
```

この方法は, find のオプション -exec rm を使用するよりも効率的な方法である. なぜなら, find の -exec オプションは該当ファイル毎にプロセスを生じさせるからである.

なお, その他の例については, 4.1.3項 (パスワードファイル (Password file) の検索例) も参照されたい.

6.6 サンプル・スクリプト (Samples scripts)

このテキストで解説したサンプル・スプリプトは, 情報科学センターの教育用マシンの以下のディレクトリに格納されているので適宜利用されたい.

```
% ls -alg /usr/ext/examples/perl/pr5/*
```

7 オンライン・マニュアル (Online manual)

Perl の関数と文法についてのオンライン・マニュアルの参照方法として，概略を表示する「エッセンス (Essence) 表示」と，詳細を表示する「フル (Full) 表示」の 2 つの方法がある．

7.1 エッセンス表示

エッセンス表示を行なう場合は，pref コマンド[16]を以下のように使用する．

```
% pref 関数名                <-- 指定した関数の解説を表示する．
% pref -l キーワード         <-- 指定したキーワード (Keywords) で始まる関数群名を表示する．
```

pref コマンドを使用して substr 関数について表示[17]した例を以下に示す．

```
% pref substr
substr(EXPR,OFFSET,LEN)
substr(EXPR,OFFSET)
        EXPR から文字列を取り出し返す。1 文字目はオフセット 0 であるが、
        $[ にセットすることで変えることができる。OFFSET が負の場合は、
        文字列の終わりからその数分戻った所から始める。

        LEN を省略すると、文字列の残り全部を返す。

        substr() 関数は左辺値として使うこともできるが、この場合 EXPR
        が 左辺値でなくてはならない。LEN よりも短いものを代入すると、
        その文字列は短くなり、LEN よりも長いものを代入するとそれに合わ
        せて長くなる。文字列の長さを変えない場合は sprintf を用いて
        pad なり chop をしなければならない。
%
```

関数名が不明確な場合は，-l オプション（l は L の小文字）に続けて関数名の先頭から数文字を指定することで，指定した先頭文字で始まる関数の一覧が表示される．以下に "ch" で始まる関数を指定した例を示す．

```
% pref -l ch
chdir
chmod
chop
chown
chroot
%
```

[16]教育用マシン /usr/ext/bin/pref． pref と /usr/ext/doc/perl/perl.dvi の著作者は次の通り．
高尾 直弥 氏， ntp@tf2.isl.mei.co.jp，松下電器産業 (株) 情報通信研究センター 情報通信関西研究所

[17]より詳細な解説を参照したい場合は，次項 7.2 (フル表示 (Xdvi)) の方法を使用されたい．

7.2 フル表示 (xdvi)

xdvi (DVI previewer) を使用してフル表示を行なう. xdvi による表示を X 端末で行なうことはできるが, パソコン等の tty 端末では行なえない.

7.2.1 通常の方法

(1) オンライン・マニュアル・ファイル (dvi ファイル) を指定して xdvi を動作させる.

教育用マシンでのファイルは /usr/ext/doc/perl/perl.dvi[15] にある. 実行例を以下に示す.

```
% xdvi /usr/ext/doc/perl/perl.dvi
```

(2) xdvi を操作してオンライン・マニュアルの内容を参照する.

メニューボタン (Menu button) が表示されるので, ボタンをマウス (Mouse) でクリック (Click) して操作を行なう. Quit のボタンをマウスで選択すると終了する.

7.2.2 Rmote (遠隔) login の場合の事前設定

Remoto login 先の Window で xdvi を使用して表示させたい場合は, あらかじめ, 以下の設定を行なっておく必要がある.

(1) 端末名を調べる.

実際に使用している端末の端末名を調べる. 情報科学センターの教育用マシンの場合は, 端末の本体前面に端末名が記載されている[18].

(2) 初期 Window での操作.

"xhost " コマンドの引数として, "+" に続けて, 遠隔マシンの正式ホスト名を入力する. 遠隔マシンのホスト名として tsuru.isci.kyutech.ac.jp を指定した例を以下に示す.

```
% xhost +tsuru.isci.kyutech.ac.jp
tsuru.isci.kyutech.ac.jp being added to access control list
%
```

(1) 遠隔 Window での操作.

遠隔マシンにログイン (Login) した後, その Window に, "setenv DISPLAY " に続いて, 実際に使用している端末名[19]の末尾に ":0" を付加した文字列を入力する.

端末名として, kamome-03.isci.kyutech.ac.jp を指定した例を以下に示す.

```
% setenv DISPLAY kamome-03.isci.kyutech.ac.jp:0
%
```

[18] 記載された端末名と, 初期 Windnow において who コマンドで表示される端末名を照合しておくとより確実である.

[19] 存在しない端末名を指定した場合は, xdvi コマンドを実行した際にエラーメッセージが表示される.

7.3 印刷

Perl のオンライン・マニュアル[20]はフリー・ドキュメントであり，GNU の General Public Licenses に従って印刷配布が行なえる（財産権の放棄）が，著作権は著作者に帰存している．このドキュメント (Document) のページ数は目次を含めて 152 ページ（A4）である．このドキュメントを Postscript printer で印刷しようとする場合は，Postscript printer を使用している一般利用者の数が少ない時間帯に行なって頂きたい．その理由は，このドキュメントを印刷するのに，15 分間程プリンターを占有してしまうからである．なお，Postscript printer の能力の関係などから，約 50 ページ以上のサイズのファイルの印刷はできない．また，一般教育環境の ID では，一度に印刷できる枚数は 30 ページまでに限定されているため，印刷出力する場合は 30 ページ以下に分割して出力する必要がある．

8 あとがき

Perl の初級レベルのスクリプトを提示して，その解説の中で必要最低限度の文法と関数の説明を行なった．この解説記事に掲載されたスクリプトを理解された方は，Perl の大半を修得したことになろう． Perl ではパターンマッチングに正規表現を使用することができるが，この解説記事では補足説明にとどめ，詳しい解説を行なわなかった．これについては，awk , sed , *grep* などに関する書籍（参考文献：その他，の項目に紹介）を参考されたい． なお，漢字コード処理，索引の自動生成，ネットワーク・インターフェイス (Network interface) を使用した例などについては解説しなかったので次の機会に解説したい．

[20]紙面に印刷されたマニュアルを，情報科学センターで閲覧することができる．

9 参考文献

UNIX 系のプログラム言語の知識を有する方は，以下に紹介した文法書といくつかの例題があれば，Perl プログラミングを易しく感じるであろう．UNIX 系のプログラム言語の知識が全くない方は，Perl の入門レベルの文献を読むか，あるいは，awk に関する書籍を参考にされたい．

【文法書】（オンライン・フリー・ドキュメント）

[1] 高尾 直弥 「日本語 perl texinfo バージョン 3.44.1」，(1991).

【解説書】

[2] WALL, LARRY. (SCHWARTZ, RANDAL L.): PROGRAMMING PERL
(NUTSHELL HANDBOOKS.), O'REILLY/SEBASTOPOL (1991)

【入門】

歌代 和正 「Little Perl Parlor」（「UNIX Magazine」），
[3] 1992.5, pp.142-147. ((1) 概説，起動法，変数，配列と添字)
[4] 1992.6, pp.150-157. ((2) 算術演算と制御構造，数値比較と論理演算，条件文)
[5] 1992.7, pp.149-155. ((3) リスト処理，配列変数操作)
[6] 1992.8, pp.157-162. ((4) 文字列処理，フィルタ)
[7] 1992.9, pp.153-159. ((5) 連想配列処理，環境変数の操作)
[8] 1992.10,pp.142-149. ((6) パターンマッチング，正規表現，多次元配列)

Rob Kolstad 「DAEMONS & DRAGONS スーパー言語 Perl」（「UNIX Magazine」），
[9] 1991.3, pp.67-73. ((1) 概説，変数と配列，フロー制御，入出力)
[10] 1991.4, pp.97-101. ((2) 正規表現，入出力，サブプログラムと関数)
[11] 1991.5, pp.86-95. ((3) 配列とベクトル関数，連想配列，システム・コール)

【初級】

serlcah@sra.co.jp 「Little Language Perl 入門」（「UNIX Magazine」），
[12] 1989.4, pp.112-117. ((3) Perl デバッガ利用例，華氏・摂氏変換例)
[13] 1989.12,pp.91-97. ((5) 引数の参照渡し，イメージダンプ，
バイナリデータ入出力，パッケージ，データベース，
システム・インターフェイス，
ネットワーク・ソケット・インターフェイス例)
[14] 1991.8, pp.119-125. ((8) 特殊文字，変数，文字列操作，漢字コード，
カタカナ単語使用場所検索例)

【初・中級】

[15] 古川 徹生 「perl - 今もっともトレンディな言語」（『ワークステーション利用の手引』九州工業大学 情報工学部 制御システム工学科編），pp.103-106 (1991).

(概説，ファイル操作，引数，オンライン・マニュアル整形表示例)

[16] serlcah@sra.co.jp 「Little Language Perl 入門」（「UNIX Magazine」），
1990.10,pp.125-130.（(6) ログイン状況チャート表示，ファイル・オーナー変更例)

[17] 前田 薫 「Perl（文字列，ファイル，プロセス処理）」（「Computer Today」）
No.50, pp.28-36.（Perl 概観，いつ Perl を使うか.）

【上級】

serlcah@sra.co.jp 「Little Language Perl 入門」（「UNIX Magazine」），
[18] 1989.8, pp.90-97.（(4) 自動マウント・コマンド例)
[19] 1991.3, pp.74-81.（(7) sendmail コンタクト・プログラム例)

serlcah@sra.co.jp 「Little Language 自由きままに Perl プログラミング」
（「UNIX Magazine」），
[20] 1990.3, pp.104-110.（(1) 属性つきファイル・アーカイバ処理への応用例)
[21] 1990.5, pp.126-132.（(2) 電話帳と住所録処理のスクリプト例)
[22] 1990.7, pp.123-130.（(3) ニュース・リーダーの作成例)
[23] 1991.5, pp.96-104.（(4) オブジェクト指向処理への応用例)
[24] 1991.10,pp.113-123.（(5) ネットワーク・ソケット・インターフェイス)
[25] 1991.12,pp.88-94.（(6) tar(梱包) ファイル内容のエッセンス参照処理例)
[26] 1992.2, pp.53-69.（(7) ネットワーク・お弁当注文受けスクリプト)
[27] 1992.4, pp.77-85.（(8) 漢字コード変換処理への応用例)
[28] 1992.6, pp.96-105.（(9) Xウィンドウへのアクセス例)
[29] 1992.10,pp.84-94.（(10) annonymous uucp の file 選択スクリプト)

【その他】

[30] A.V. エイホ／B.W. カーニハン／P.J. ワインバーガー（足立高徳 訳）
「プログラミング言語 AWK」，トッパン (1989).

[31] Dale Dougherty（福崎俊博 訳）「sed & awk」，アスキー出版局 (1991).

[32] 今泉 貴史 「UNIX 流プログラミング」（「UNIX Magazine」）
1992.1, pp.123-127.（(15) 言語の選択基準：sed,awk,sh,csh,C 比較)

補足目次