AN_HTTPD サーバ*と ActivePerl†ではじめる CGI アプリケーション

千葉商科大学付属高等学校 数学科 樽　正人

2005 年 1 月 5 日

1　プログラムの準備

1.1　AN_HTTPD の準備

1.1.1　AN_HTTPD とは

- AN_HTTPD の現在の最新バージョンは 1.42 になります。
- AN_HTTPD は，パソコンを WEB サーバとして稼動させることができます。
- 例えば，AN_HTTPD が稼動しているパソコンをⒶとします。Ⓐがインターネットにつながっていれば，インターネット上の任意のパソコンからⒶ**に保存してある HTML ページ**をホームページとして公開できます。
- AN_HTTPD はインターネット上に無償で配布されているフリーソフトウエアです。
- このサイトのアドレス（URL）は，`http://www.st.rim.or.jp/%7Enakata/` になります。
- インストール方法や環境設定など詳しい情報が上記サイトから調べることができます。

1.1.2　AN_HTTPD のインストール

1. まず最初に，AN_HTTPD の圧縮ファイルを上記サイトからダウンロードします。
2. 保存先の準備として，デスクトップの [マイコンピュータ] から H:ドライブにアクセスして，“httpd” という名前のフォルダを新規に作成しておきます。
3. 今回は，すでにインターネットから httpd142n.zip という圧縮ファイルを学内サーバにダウンロードしてあるので，以下の URL からダウンロードします。

   `http://alice/~scorpion/cgi-study/httpd/httpd142n.exe`
4. ダウンロードしたファイル httpd142n.exe は，自己解凍型プログラムになっているので，ダブルクリックして解凍します。
5. 現れた英語のポップアップ画面で，“Unzip” ボタンをクリックします。
6. 解凍が終了したら，先ほどのポップアップ画面で，“close” ボタンをクリックして終了します。
7. `H:\httpd` をマイコンピュータから開くと，必要なプログラムが解凍されていることが確認できます（アイコンは 16 個あります）。

1.1.3　AN_HTTPD の設定

1. AN_HTTPD は多機能な大変優れたソフトウエアです。詳しい設定は `H:\httpd` の中の，“readme.html” を参照しましょう。

---

* http://www.st.rim.or.jp/%7Enakata/

† http://www.activestate.com/Products/ActivePerl/

2. ここでは，WEB サーバとして稼動するための必要最低限の設定のみを説明します。
3. H:\httpd の中の，httpd.exe をダブルクリックで起動します。
4. すると右下のタスクトレイに常駐ソフトとしてアイコン化されますので，
   そのアイコンを右クリックして，“オプション一般（G)...” を選択します。
5. 図 1 のようなポップアップ画面が現れるので，” ドキュメントルート” の設定をします。ドキュメントルートとは，HTML ファイル（ドキュメント）を置く場所の基点（ルート）を指します。すなわち，インターネット上で公開したいファイルの保存場所の設定です。ここでは，H:\web_local にします。図 1 を参考にして，2 箇所のチェックボックスを確認したら，最後に画面下にある拡張子.pl,.cgi の行を選択し，編集ボタンをクリックして実行プログラムの欄を c:\perl\bin\perl.exe に変更します。

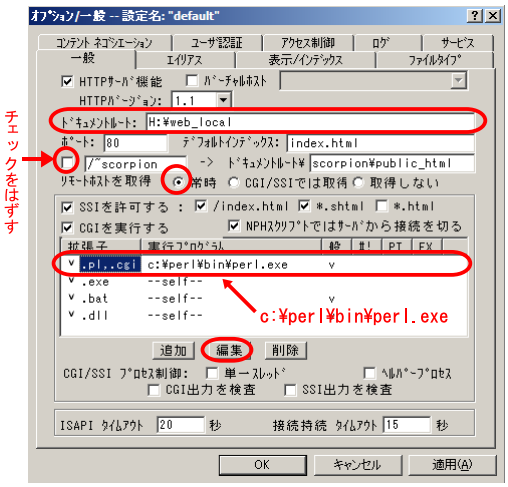

図 1　環境設定『一般』

6. 次にエイリアス[*1]タブをクリックして，cgi アプリケーションの仮想パスを図 2 のように
   H:\web_local\cgi-bin [*2]に変更します。また，デフォルトの script の設定は削除しておきます。

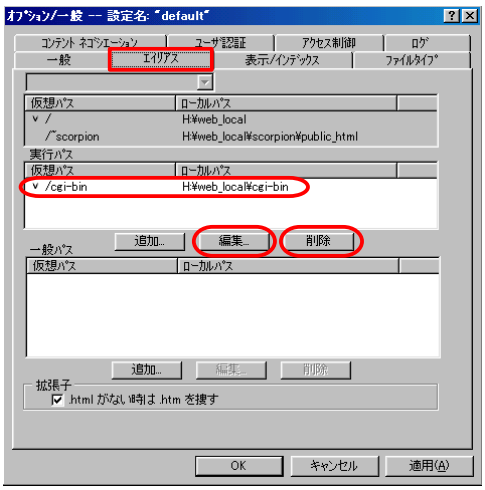

図 2　環境設定『エイリアス』

7. 以上の設定が終了したら，OK ボタンをクリックします。

[*1] エイリアスとは別名という意味
[*2] 実は AN_HTTPD の現バージョンでは，ネットワークドライブをエイリアスに指定することができません。
対処療法として\\alice**ユーザー**\web_local\cgi-bin とします。

1.2 ActivePerl の準備

1.2.1 ActivePerl とは

- ActivePerl の最新バージョンは，5.8.6 になります。
- ActivePerl は，UNIX の世界では必須なプログラム言語 Perl を Windows に移植したものです。
  特にテキストファイルの入出力や検索，抽出などに長けた言語で，WEB サーバを介した入出力を中心とする CGI アプリケーションの開発言語として知られています。
- 皆さんが良く利用するインターネット上の Chat や掲示板などを開発する言語といえばピンとくるかもしれません。
- ActivePerl はインターネット上に無償で配布されるフリーソフトウエアです。
- サイトアドレス（URL）は，http://www.activestate.com/ になります。

1.2.2 ActivePerl のインストール

1. `http://www.activestate.com/Products/Download/Download.plex?id=ActivePerl` から，Windows 版の最新バージョンを適当なフォルダにダウンロードします。たとえば，先ほど AN_HTTPD を保存した H:httpd でもいいでしょう。
2. ダウンロードしたファイルをダブルクリックしてインストールを開始します。
3. 以下のサイトに詳しいインストール方法が記載されています。
   http://greenforest.zive.net/server7.html
4. 学校では，一つ前のバージョン 5.6.1 がすでにインストールされていますので割愛します。

1.3 AN_HTTPD と ActivePerl の動作確認

1.3.1 AN_HTTPD(WEB サーバ) の確認

1. WEB サーバとしての稼動を確認します。
2. たとえば alice という WEB サーバを利用している h0110999 というユーザーの index.html をホームページとして確認する場合，`http://alice/~h0110999/` とサーバ名とユーザー名の指定が必要でした。
   しかし，自分のパソコンに AN_HTTPD が WEB サーバとして稼動している場合，自分のパソコンがサーバになるので，サーバ名は localhost に，また `H:\web_local` をドキュメントルートにしたためユーザー名の指定もいらなくなります。（下図を参照）

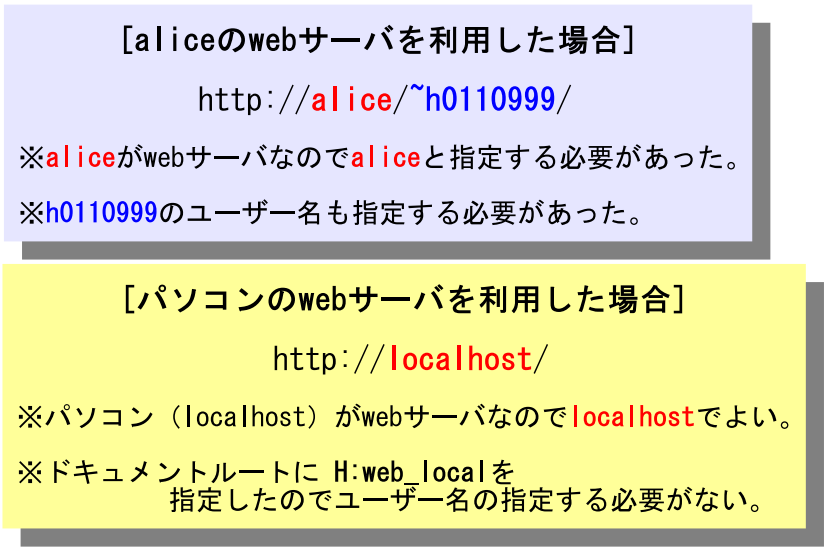

図 3　稼動する web サーバの違い

1.3.2 ActivePerl(CGI アプリケーション) の確認

1. CGI アプリケーションとしての動作を確認します。
2. xyzzy というエディタを起動して，以下の内容を入力します。

```
#!/usr/bin/perl

print␣"Content-type:␣text/html\n\n";
print␣"<html>\n";
print␣"<body>\n";
print␣"<h2>Hello␣World</h2>\n";
print␣"</body>\n";
print␣"</html>";
```

（注意）\ は ¥ 記号で，␣ は半角スペースのこと

3. H:\web_local\cgi-bin に，”hello.cgi” という名前で保存します。
4. ブラウザから URL を，http://localhost/cgi-bin/hello.cgi に指定して開きます。以下の画面が出力されれば成功です。

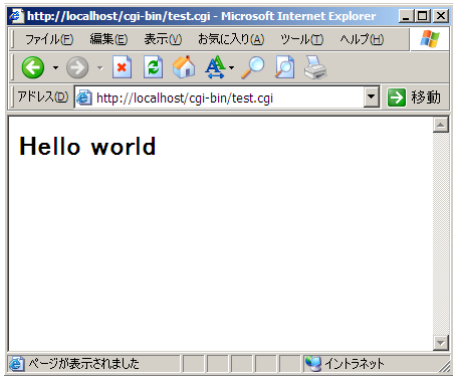

図 4　test.cgi

5. 上記の Perl ソースは，html を出力する簡単な CGI アプリケーションになります。

2　TCP/IP プロトコルの基本

2.1　TCP/IP は５階層構造

現在のインターネットを支えているデータ転送技術が TCP/IP です。

たとえば，自宅のパソコン ⓒから，千葉商科大学付属高等学校の WEB サーバ ⓢにアクセスしているとき，表面的には自宅と高校の 2 点間でデータ通信をしているように見えます。

ところが，実際はその間にいくつものサーバが介在し，それらを経由しながらデータは転送されています。

このように表面的には見えていないが，水面下では複数の階層がそれぞれに仕事を担っている様を **透過的** といいます。

TCP/IP は５つの階層（レイヤー）で構成され，透過的に 2 点間通信を実現させています。

アプリケーション層（HTTP，FTP など）
トランスポート層（TCP/UDP）
ネットワーク層（IP）
データリンク層（イーサーネットの規格）
物理層（コネクタの形状，ケーブルの規格など）

表 1　TCP/IP の階層

2.1.1　データのカプセル化

例えば，インターネットエクスプローラが指定した WEB ページを表示するためには，最上位層のアプリケーション層を利用します。そのためユーザーは表面的には 2 点間でデータ通信をしているように見えます。しかし，実際にはアプリケーション層ではリクエストのあったデータをすぐ下のトランスポート層に渡します。データを受け取ったトランスポート層では，まずそのデータをカプセル化します。次に頭（ヘッダー）の部分にトランスポート層同士の通信で必要になる情報を付加して，またすぐ下のネットワーク層に渡します。

同じことがその下の層に対しても続いていくのですが，これ以降のネットワーク層，データリンク層，物理層からは，インターネット上の相手先にデータを転送するために必要な情報がヘッダー部分に付加されます。したがってインターネット上を経由するコンピュータには，この下位３層のデータを理解できることがもとめられます。

では，この３層のヘッダー部分にはどういった内容が付加されていくのかというと，IP アドレスの番号や ADSL 回線や電話回線等の規格に関する情報や光ケーブルやメタケーブル類の情報だけです。ですから，中継するコンピュータが Windows，Macintosh，Linux，FreeBSD 等々，どんなＯＳで動いていようが関係ないのです。**これが TCP/IP が何層にも分けてデータをリレーする理由**になります。

すなわちインターネット越しにデータを取得したい，あるいは配信したいと考えたとき，単にデータの送受信だけであるならば，ＯＳ等の機種に依存することなく TCP/IP のデータのカプセル化とそのリレーという技術によって可能になるのです。もちろん送受信したデータそのものを開くためには，お互いに作成したソフトウエアを持っていることが必要になるのですが。

2.1.2　データの暗号化

このようにデータは，インターネット上を経由する際にカプセル化されています。ですから中継するコンピュータ上で，中身を知ることは出来ません。しかし，意図せず転送中にカプセルが壊われたり，あるいは意図的に空けて中身を傍受されたり，改ざんされたりすることはあるかもしれません。これがインターネットの弱点でもあります。

そこで，仮にカプセルの中身を傍受されても，その情報自体を暗号化しておけば，解釈はされませんし，改ざんしようとしても，最悪データ自体が壊れるだけ済みます。

例を上げると，メールを閲覧する際に流すパスワードなどがあります。このパスワードはそのまま[*3]カプセル化してメールサーバに転送されます。このままでは，意図すればインターネット上で簡単に中身を確認されてしまいます。
しかし，カプセル化する前にこのパスワードを暗号化します。そうすれば，仮にカプセルの中身を開かれても，復号化するための鍵を持っていなければまったく内容を確認されることはありません。

2.2 トランスポート層＝ポート

実は，アプリケーション層ではインターネットエクスプローラだけでなく，メール送信プログラムなど，その他のプログラムも同時に稼動しています。このように多重にプログラムを稼動できる環境をマルチスレッド環境といいます。

このようにそれぞれ異なるプログラムからの異なるデータをトランスポート層は，混乱なく受け取らなければなりません。そのためアプリケーション層とトランスポート層で複数のデータを区別してやり取りする必要があります。それがポート（港）です。ポートは 16 ビット，すなわち 2 の 16 乗個の番号が用意されています。これは，データを港で積み込みますが，１つの港では同じ種類のデータしか積み込めません。種類ごとに空いている港を見つけ出して，積み込みするイメージになります。

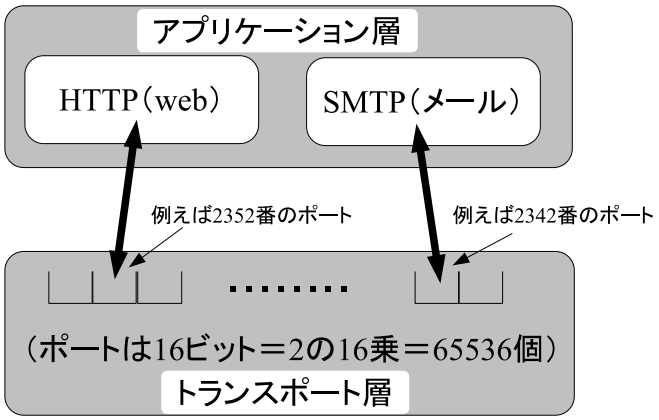

図 5　ポートによる複数送信

2.3 ウェルノウンポート

ウェルノウンポート（well-known）とは，特権ポートなどとも呼ばれる 1〜1023 番までの予約されたポートを言います。

インターネットでの通信は，基本的にクライアントとサーバ間で行われます。サーバを特定するには IP アドレスを指定すればよいです。しかし，前節 2.2 のようにコンピュータでは複数のプログラムが稼動していて，それぞれがポートを使って通信しています。これはサーバも同じです。ですから，IP アドレスの指定だけでは利用したいプログラムまで特定することができません。

実は，インターネット上で通信を確立するには，IP アドレス＋ポート番号の組合わせが必要になるのです。
ところが受けてのサーバが，前述のクライアントのように空いているポートを自由に使用してしまうと，クライアントはサーバのどのポートがどんなプログラムのサービスになっているかを知ることが出来ません。どのポートにリクエストを送ればよいのかわからないわけです。

そこで，各プログラムがそれぞれ何番のポートで待機しているかをあらかじめ決めておく必要があるわけです。このあらかじめ予約したポートを well-known ポートといいます。代表的なポート番号を表 2 に示します。

[*3] 平文（ひらぶん）といいます。

サービス名	ポート番号
FTP-DATA	20
FTP(control)	21
SSH	22
TELNET	23
SMTP	25
DNS	53
HTTP	80
POP3	110
IMAP	143

表 2　主な well-known ポート

以上のことから，千葉商科大学付属高等学校のホームページを閲覧したいときは,

1. ホームページ形式である　→　http
2. 千葉商科大学付属高校の URL　→　www.hs.cuc.ac.jp
3. ホームページが使うポート　→　80
4. ドキュメントルートにある index.html　→　index.html

上記４つの情報を組み合わせてはじめて取得したいデータを特定できます。

http://**www.hs.cuc.ac.jp**:**80**/**index.html**

一般にはポート番号 80 やトップページ index.html を省略して,

http://www.hs.cuc.ac.jp/ になります。

3 Perl を学ぼう

3.1 準備編

3.1.1 Perl の概要

IT用語辞典 e-Words より

Perl[a]は，Larry Wall 氏が開発したプログラミング言語。テキストの検索や抽出、レポート作成に向いた言語で、表記法は C 言語に似ている。インタプリタ型であるため、プログラムを作成したら、コンパイルなどの処理を行なうことなく、すぐに実行することができる。CGI の開発によく使われる。とにかく機能が豊富なことで知られる。当初は UNIX 上で利用されたが、現在では Windows を含む様々なプラットフォームに移植されている。

[a] Practical Extraction and Report Language の頭文字

■インタプリタ型とコンパイル型　プログラム言語には，Perl のように記述したソースを実行しながら同時に解釈をしていく インタプリタ型 と，C 言語のように，記述したソースを先に機械にわかる形に直してから実行する コンパイル型 の 2 つに分類されます。

コンパイル型　訂正のたびにコンパイルをしなおす手間がありますが，実行速度が向上します。

インタプリタ型　訂正後すぐに実行できる手軽さがありますが，実行速度は劣ります。

■Perl に出来ること　プログラム言語ですから，ほぼ出来ないことはありません。特にネットワークコンピュータを制御する UNIX に標準装備され，ネットワーク上のデータを加工したり，作成したりすることができます。そのため WEB 上に見られる掲示板や Chat などのプログラムを開発する言語として定番になっています。

3.1.2 Perl の基本構文

エディタにプログラムを記述することからはじめますが，

- 最初の行にはプログラムの本体である Perl が置いてある場所[*4]を指定します。
  #! の記号からはじまります。
- 次の行から，実際にプログラムを記述することになります。１行ごとに処理を１つずつ記述しますが，必ず行末にデリミタ[*5]として; をいれます。
- #は，プログラムとして実行させたくない部分の頭に付ける特殊な記号です。

次に例を示します。これは画面に”Hello World” とだけ表示するプログラムです。

```
#!c:\perl\bin\perl.exe   # 最初にパスをこのように指定します
print "Hello World";   # 最後にデリミタの; を忘れません。
```

構文として守らなければいけないことは，これだけです。

[*4] path（パス）といいます。一般に UNIX 環境では，usr 内にある bin というフォルダの中に，perl という名前でおかれています。
[*5] 区切りという意味。

3.1.3 Perl を学ぶための環境準備

1. UNIX から，Windows 用に移植された ActivePerl という Perl を使用します。これは，すでに学校のコンピュータにインストールされています。
2. 作成したプログラムを保存する場所を作ります。 H:ドライブに，perl-study という名前のフォルダを新規に作成します。
3. プログラムを記述するために使用するエディタとして xyzzy を選びます。
4. Perl は Windows のプログラムと違い，コンソール[*6]上で実行されます。Windows 用のコンソールとして，
   ”コマンドプロンプト ”が用意されています。

   スタートメニュー → すべてのプログラム → アクセサリ → コンマンドプロンプト

3.1.4 作業の流れ

1. xyzzy を使って，Perl のプログラムソースを記述します。
2. “ファイル名.pl” という形式で，先程作成した `H:\perl-study` に保存します。

   例えば，hello.pl という感じになります。

3. コンソールを起動したら，`H:\perl` に移動[*7]します。次にソースファイルを指定して Perl を実行します。

   （ 例えば上記の例で，hello.pl を実行する場合）
   ```
   H:\>cd perl-study     # perl-study に移動
   H:\PERL-STUDY>perl hello.pl     # hello.pl を実行
   ```

4. コンソール画面に，実行結果が表示されれば成功です。

3.1.5 コンソールの基本操作

実行にはコンソールでの操作が必要になります。ここでよく使うのコマンドを紹介します。

コマンド名	コマンドの説明
`dir␣[/p]`	現在のフォルダにあるファイルの一覧を出力します。一画面ずつ表示したい場合は，p オプションを与えて実行します。
`cd␣`**<フォルダA>**	フォルダAに移動します。
`mkdir␣`**<フォルダA>**	新規にフォルダAを作成します。
`copy␣`**<ファイルA>**`␣`**<ファイルB>**	ファイルAをファイルBという名前でコピーします。
`del␣`**<ファイルA>**	ファイルAを永遠に削除します。

表 3　よく使うコンソールコマンド一覧

練　習　実際にコンソールを起動して，上記のコマンドを実行してみましょう。

[*6] 直接キーボードからコマンドを打ち込んでいくために用意された原始的な画面

[*7] コンソールのコマンド cd を使用します。

3.2 基礎編

3.2.1 Hello World の世界

1. P9 の 3.1.4 節の手順で xyzzy を起動したら，以下の内容を入力します。

```
#!c:\perl\bin\perl.exe

print ‘‘Hello World’’;
```

2. H:\perl-study に，hello.pl という名前で保存します。
3. コンソールを起動し，H:\perl-study に移動したら， perl hello.pl と実行します。
4. コンソールに Hello World と出力されたら成功です。

練習　上記のソースを改変して，

```
Hello World
Good Bye
```

と 2 行のメッセージがコンソールに出力するようにしましょう。
ヒント　改行命令は, \n

3.2.2 スカラー変数

Perl のすばらしさは，変数を自由に扱える点です[*8]。変数とは，数学の文字で言うと x にあたります。いわばデータを格納する箱です。

スカラー変数には 1 度に 1 つのデータしか入りません。2 つも 3 つも入れる箱にはなれません。[*9]

スカラー変数は，スカラー変数の証として先頭に$をつけます。また，変数名として使える文字は，半角英ではじまる半角英数字と記号_（アンダーバー）だけです。数字からはじまる名前は使えません。

よい例　$abc　や　$var1	悪い例　$123　や　$1var

スカラー変数を使ったプログラムの例を下に示します。

```
1 行目　#!c:\perl\bin\perl.exe                    ↓↓実行結果↓↓
2 行目　$a = 3;
3 行目　$b = 5;
4 行目　$ans = ’答え’;
5 行目　print ‘‘スカラー変数のサンプル１\n\n‘‘;
6 行目　print ‘‘$a + $b の$ans は，‘‘;
7 行目　print $a + $b;
```

実際に入力して実行する前に，右の余白に予測結果を書いてみよう。
予測したら，“var1.pl” という名前で保存し，実行結果を確認してみよう。

[*8] Perl に限らずプログラム言語はどれも変数を扱えますが，プリミティブな言語ほどその扱いは厳密になります。変数の大きさを間違えるとバッファオーバーフローといってメモリからデータがあふれでる重篤なエラーを引き起こす原因になるからです。

[*9] 配列変数といいます。後述します。

変数$a と$b には数値（シングルクオートなし）が，変数$ans には文字列（シングルクオートあり）が代入されています。また，6 行目と 7 行目の + の処理の違いに注意しましょう。

■シングルクオート（'）とダブルクオート（"）の違い　スカラー変数（以降，変数という）に文字列データを格納する際[*10]は，必ず シングルクオート か ダブルクオート でその文字列を囲む（くくるという）のですが，そのデータ中に変数が含まれているとき，

- シングルクオートでくくった場合の変数は展開されません。

例）

```
#!c:\perl\bin\perl.exe
$a = 3;
$b = 5;
print '$a と $b は展開されません';
```

（結果）

```
$a と $b は展開されません
```

- ダブルクオートでくくった場合の変数は展開されます。

例）

```
#!c:\perl\bin\perl.exe
$a = 3;
$b = 5;
print ``$a と $b は展開されます'';
```

（結果）

```
3 と 5 は展開されます
```

練習　先程の var1.pl ソースを使用して，実際に確認してみましょう。

3.2.3 演算子の種類

演算子には，**２項演算子**と**単項演算子**の２種類があります。

■２項演算子　２つの値を演算する記号[*11]のことです。次のようなものが用意されています。

演算子	解説
+	加法（例　3+2 → 5）
-	減法（例　3-2 → 1）
*	乗法（例　3*2 → 6）
/	除法（例　6/3 → 2）
**	べき乗（例　3**2 → 3 の 2 乗）
%	剰余（例　12%5 → 12 を 5 で割った余り）
.	文字列連結（例　" あいう"." えお" → あいうえお）

■単項演算子　１つの値を対象に演算した結果を代入していく特殊な演算記号です。コンピュータ独自の仕様です。

演算子	解説
++	左辺（右辺）に 1 を加えた結果を，左辺（右辺）に代入する（例　$i++　(++$i)）
−−	左辺（右辺）から 1 を引いた結果を，左辺（右辺）に代入する（例　$i−−　(−−$i)）

[*10] var1.pl の 7 行目のように文字列ではなく，演算処理をさせたい場合にはクオートでくくってはいけません。
[*11] 例）3 + 2 など

単項演算子が，前（++$i）にある場合は，その場で 1 を加えた値が代入されるので，

（例） **（結果）**

```
$i = 0;                          1
print ++$i;
```

となります。
それに対して，後ろ（$i++）にある場合は，代入が遅れますので，

（例） **（結果）**

```
$i = 0;                          0
print $i++;
```

となります。この違いだけは注意しましょう。

■演習問題

1. $name1 という変数に自分の苗字を，$name2 という変数に自分の名前を代入し，文字列連結演算子を利用して，フルネームを画面に出力してみよう。

```
#!c:\perl\bin\perl.exe
$name1 = "山田";
$name2 = "太郎";

print （                    ）;
```

2. 次のカッコ内に適する数式を入れ，プログラムを完成させよう。

```
#!c:\perl\bin\perl.exe
$a = 2;
$n = 16;

# 画面に 2 の 16 乗の結果を出力する。
print "2 の 16 乗の答えは，".（                ）."\n";
```

3. 単項演算子（++） を利用して，次のカッコに適する数式を入れよう。

```
#!c:\perl\bin\perl.exe
$year = 2005;

# 画面に「今年は 2005 年だから，来年は 2006 年ですね。」と出力させる。
print "今年は，$year 年だから，来年は（          ）年ですね。\n";
```

3.2.4 配列変数

3.2.2 節のスカラー変数には，一度に１つの値しか代入できませんでした。
それに対し，配列変数では一度に複数の変数を扱うことができます。
いわば，スカラー変数を 1 列に複数並べたイメージになります。

配列変数

スカラー変数１，スカラー変数２，スカラー変数３，...

配列変数は，配列変数の証として先頭に@マークをつけます。
例）`@array` **や** `@pairs`

■配列変数への代入　配列変数に値を代入するには，次のように値をカンマで（,）で区切り，カッコで囲みます。

```
@array = ('apple','orange','melon');
```

■配列変数から値の参照　配列変数のそれぞれの変数は，添え字と呼ばれる 0 から始まる整数番号で識別されます。上記の例の場合，"apple"の値が代入されている変数の添え字番号は 0 になります。
ですから，値を代入する方法として

```
$array[0]*12 = 'apple';
$array[1] = 'orange';
$array[2] = 'melon';
```

というのもあります。
また，逆に代入した値を参照したい場合は

例）	（結果）
`print $array[0];`	apple

となります。

■練習　次のカッコ内に適するものを入れよ。

```
#!c:\perl\bin\perl.exe

# 配列変数@weeks に，'日','月',...,'土'を代入する。
(                                        );

# 今日の曜日を画面に出力させる。
print "今日は，(          ) 曜日です。\n";
```

*12 先頭が@でなく，スカラー変数と同じ$である点に注意。

3.2.5 連想配列変数（ハッシュ変数）

（キーワードで）連想（できる）配列変数という意味になります。配列変数では，添え字を頼りにデータを取り出しましたが，連想配列変数では，添え字の変わりにキーワードを利用できます。連想配列変数は，その証として先頭に % マークをつけます。

例）`%hash` **や** `%form`

■連想配列変数（ハッシュ変数）への代入　代入方法は２つありまが，いずれもキーワードとデータの対（ペア）で代入します。

その１）

```
%hash = ('name','Jane','country','America','sex','female');
```

その２）

```
$hash{'name'} = 'Jane';
$hash{'country'} = 'America';
$hash{'sex'} = 'female';
```

どちらも name というキーで Jane という値を，country というキーで America という値を，sex というキーで female という値を連想配列変数（ハッシュ変数）%hash に代入したことになります。

■連想配列変数（ハッシュ変数）から値の参照　配列変数と違い，欲しい値はキーとして定義した言葉から容易に参照できます。

例）
ハッシュ変数 %hash から Jane という値を取り出すには，`$hash{'name'}`[*13]とします。

```
%hash = ('name','Jane','country','America','sex','female');
print "$hash{'name'}\n";
```

上の場合，画面には Jane と出力されます。

■練習　ハッシュ変数 %foods について，

（１）meat をキーで cow という値，drink をキーで milk という値，soup をキーで corn という値を代入し，

（２）milk という値を画面に出力する

プログラムを hash_ex.pl という名前で作成しなさい。

[*13] 先頭が % でなく，スカラー変数と同じ$である点に注意。

3.3 実践編

3.3.1 条件分岐 if と else 文

条件を事前に用意しておき，その条件を満たす時と満たさない時で処理内容を変更するときに if 文を使用します。文法は次のようになります。

```
if(条件){
条件を満たす時の処理
}
else{
条件を満たさない時の処理
}
```

例）変数$num の値が 1 ならば成功と画面に出力。それ以外なら失敗と出力。

```
$num = 1 # ここの値を色々変えてみる。
if($num == 1){
print "成功";
}
else{
print "失敗";
}
```

ここで，条件の中に”==”という記号がありますが，このように左辺値と右辺値を比較する演算子のことを
比較演算子といいます。

■主な比較演算子の紹介　if 文を使うときに，比較演算子は必要になります。ここで主な比較演算子を紹介します。

比較演算子	解説
==	数値 1 == 数値 2 というように使います。等しければ true です。
!=	数値 1 != 数値 2 というように使います。等しくないとき true です。
eq	文字列 1 eq 文字列 2 というように使います。等しければ true です。
ne	文字列 1 ne 文字列 2 というように使います。等しくないとき true です。
>	数値 1 > 数値 2 というように使います。数値 1 の方が大きいとき true です。
>=	数値 1 >= 数値 2 というように使います。数値 1 と等しいか大きいとき true です。
<	数値 1 < 数値 2 というように使います。数値 1 との方が小さいとき true です。
<=	数値 1 <= 数値 2 というように使います。数値 1 と等しいか小さいとき true です。

3.3.2 ファイルの入出力処理　open 関数

open 関数を使えばデータをファイルに出力したり，逆にファイルからデータを入力したりすることができます。文法は次のようになります。

ファイルから入力するとき，

```
open(ファイルハンドル名[*14],"<入力（取り込む）ファイル名");
```

例）ABC.txt ファイルのデータを入力して，画面に出力。

```
open(IN,"<ABC.txt");
print <IN>;        # ファイルハンドルを直接扱うときは<>で囲む
close IN;          # 開いたファイルは，close 関数で必ず閉じること。
```

ファイルに出力するときは上書きと追書きの 2 つのモードがあり，

```
open(ファイルハンドル名,">出力（上書き）ファイル名");
open(ファイルハンドル名,">>出力（追書き）ファイル名");
```

例 1）XYZ.txt ファイルにデータを出力（上書きモード)。

```
open(OUT,">XYZ.txt");
print OUT "書き込みテスト\n";
close OUT;
```

例 2）XYZ.txt ファイルにデータを追書き出力（追書きモード)。

```
open(OUT,">>XYZ.txt");
print OUT "もう一度書き込みテスト\n";
print OUT "ここで終了。\n";
close OUT;
```

3.3.3 繰り返し処理　foreach 文

複数のデータを 1 つずつ取り出しながら，同じ処理を繰り返したい。そんな時に便利なのが，**foreach 文**です。**foreach 文**は，配列変数からデータがなくなるまで，一つずつ取り出します。文法は次のようになります。

```
foreach スカラー変数（配列変数){
      処　理;
}
#配列変数から１つずつ順に取り出したデータを指定したスカラー変数に代入します。
#代入先のスカラー変数が省力された場合は，特殊変数$_に代入されます。
```

例 1）配列変数@fruits からデータを一つずつ取り出し，画面に出力。

```
@fruits = ('apple','orange','melon','grape','strawberry');
foreach $value (@fruits){
    print "$value\n";
}
```

[*14] ファイルハンドルとは，ファイルを処理するためのオブジェクトです。一般に，**適当な半角英語の大文字**などで表します。

例 2）XYZ.txt ファイルから，データを一つずつ取り出して画面に出力。

```
# XYZ.txt からデータを入力して，配列変数@pairs に格納
 open(IN,"<XYZ.txt");
@pairs = <IN>;
 close IN;

# 配列変数@pairs からデータを１つずつ取り出す
 foreach $value (@pairs){
   print "$value\n";
}
```

3.3.4 データの分割処理　split 関数

split 関数は，指定したコード（文字）でデータを切り分けたいときに使用します。切り分けたデータは，新たに配列変数に格納したりします。文法は次のようになります。

```
配列変数 = split(/区切り文字/, データ)
#指定した区切り文字でデータを切り分けて，配列変数に格納します。
```

例）データを区切り文字"^"で切り分けて，配列変数 @parts に格納したら画面に結果を 1 行ずつ出力。

```
$line = '幼年期^若年期^成年期^中年期^老年期';
@growth = split(/^/,$line);
foreach $parts (@growth){
  print "$parts\n";
}
```

■練習問題　次のスカラー変数$buf からの右辺値だけを取り出し，一行ずつ画面に出力しよう。

```
$buf = 'var1=apple&var2=orange&var3=melon&var4=grape&var5=strawberry';
```

手順

1. &を区切り文字として split して適当な配列変数に格納する。
2. 次に格納した配列変数から foreach 文でデータを一つずつ取り出しながら，でまた split しながら，画面に出力する

3.3.5 エラーと例外処理

プログラム（コンピュータ上に限らず）に，エラーはつきものです。プログラム自体が招くエラーと周りの環境から招かれるエラーとがあります。前者の場合は，そのプログラム自体の問題ですから, その部分を改変することでエラーを無くせます。しかし，後者の場合は利用するユーザーによって環境もさまざまであるため，エラーを確実に無くすことは不可能と言えます。

そこで予期せぬエラーにおいては，期待した結果が得られないまでも，大切なデータは保護するといったような処理が必要になります。**こういった処理を例外処理といいます**。

特に前節 3.3.2 で利用した open 関数の場合，プログラムを実行する環境によっては指定したファイルを作成できなかったり，読み込みに失敗することなどがあります。open 関数は，処理

に成功した場合には true の 1 を，失敗した場合には false の 0 を返します。**これを戻り値といいます**。

open 関数に限らず，perl で使われる関数では一般に成功した場合は True の 1 を，失敗した場合には False の 0 を戻り値として返すようになっています。
この戻り値を使えば，次のような if 文を使った例外処理が可能です。

```
#!c:\perl\bin\perl.exe

if(open(OUT,">>TEST.txt") eq 0){
    print "Error\n";
    exit;
}
@lines = <IN>;
close IN;
・・・・・
```

■練習問題　前節 3.3.3 の例 2）に例外処理を加えたプログラム foreach-samp-err.pl を作成しよう。

3.3.6 サブルーチン化

4 CGI アプリケーションの作成

4.1 環境整備編

4.1.1 CGI アプリケーションの保存先

4.1.2 CGI アプリケーションの属性

4.2 基礎編

4.2.1 HTML の出力

4.2.2 フォームからデータの送信

4.3 実践編

4.3.1 フォームからデータの受信

4.3.2 データの処理

4.3.3 ファイルの出力

4.3.4 セキュリティの向上