Ｐｅｒｌの基礎Ｉ

０．目次

１．Ｐｅｒｌとは

Ｐｅｒｌ（Practical Extraction and Report Language）は、高機能を備える汎用のプログラミング言語である。インタープリタで実行されるので、Ｃ言語のように実行する前にコンパイルする必要がない。ファイル内のテキスト操作やデータの抽出などのために開発されたが、プロセスの操作やネットワーク関係の作業もできるようになってきた。エディタでＰｅｒｌプログラムを作成・修正後、すぐ実行できる。

２．Ｐｅｒｌのインストールと実行

２．１　Ｐｅｒｌのインストール

ホスト	hcs.ipc.ibaraki.ac.jp
ＦＴＰサイト	ftp.chiba-u.ac.jp
ディレクトリ	/pub/GNU/perl
ファイル	perl-5.005.02.tar.gz

①展開

（１）作業ディレクトリ（/short/isemba）の下にファイル（perl-5.005.02.tar.gz）を移動する。

```
% cd /short/isemba
% gzip -d perl-5.005.02.tar.gz
% tar -xf perl-5.005.02.tar
```

②コンパイル

（１）インストール先のディレクトリ（/short/isemba/local,/short/isemba/local/bin）を作成する。

```
% mkdir /short/isemba/local
% mkdir /short/isemba/local/bin
% cd perl-5.005.02
% ./Configure （約５分）
```

質問には、ほとんどリターンキーを押せばよいが、インストール先を聞かれたときは/usr/localを/short/isemba/localに変更する。

```
% make （約２分）
% make test （約２分）
% make install （約２分）
```

③実行

```
% /short/isemba/local/bin/perl perlファイル
```

④確認

```
% perl -v
```

２．２　Ｐｅｒｌの実行

プログラムは、エディタ（vi,muleなど）でファイル（prog.pl）に作成した後、Perlインタープリタで実行する。

```
% perl prog.pl
```

または、ファイル（prog.pl）を実行可能にして、コマンドとして実行できる。

```
% chmod 700 prog.pl
% prog.pl
```

短いプログラムの場合、-eオプションを使って、

```
% perl -e 'print"hello!¥n";'
```

とすると、

```
hello!
```

と出力される。

マニュアルは、man perl, man perlfunc, man perlop, man perlvarなどで調べる。

man perl	全体の説明
man perlfunc	関数
man perlop	演算子
man perlvar	特殊変数

３．Ｐｅｒｌの書き方

３．１　入出力

３．１．１　文字列の出力

```
1 #!/usr/local/bin/perl
2 # << p311.pl >>
3 # サンプル・プログラム
4 print"Hello, Perl!¥n";
5 print"こんにちは¥n";
6 exit(0);
```

実行結果

```
% perl p311.pl
Hello, Perl!
こんにちは
```

説明

１行目	プログラムの最初の行は、#!の後にperlの絶対パス名を指定する。
２行目	#から以降は、注釈である。メモ等を書くとよい。
４行目	print文は、printで始まり、"で囲まれた文字列が続く。¥nは改行を意味する。Perlでは、文は；で終了する。
６行目	正常終了の場合0、異常終了の場合は1-255の値を指定する。

３．１．２　数値、文字列を書式に従って出力

```
 1 #!/usr/local/bin/perl
 2 # << p312.pl >>
 3 $a = 123;
 4 printf"|%5d| |%-5d|¥n",$a,$a;
 5 $b = "abc";
 6 printf"|%5s| |%-5s|¥n",$b,$b;
 7 $c = 12.345;
 8 printf"|%8.3f| |%-8.3f|¥n",$c,$c;
 9 $d = -12.345;
10 printf"|%12.3e| |%-12.3e|¥n",$d,$d;
11 exit(0);
```

実行結果

```
% perl p312.pl
|  123| |123  |
|  abc| |abc  |
|  12.345| |12.345  |
|  -1.235e+01| |-1.235e+01  |
```

説明

３行目	実行中に変化するデータは変数に保存される。 整数、実数を表す変数は、変数名の前に「$」を付ける。
４行目	%5dは整数を５桁内に右寄せ、%-5dは整数を５桁内に左寄せ。
５行目	文字列は、「"」で囲み、文字列を表す変数は、変数名の前に「$」を付ける。
６行目	%5sは文字列を５桁内に右寄せ、%-5sは文字列を５桁内に 左寄せ。
８行目	%8.3f実数を８桁内（小数点以下３桁）に右寄せ、%-8.3fは 実数を８桁内（小数点以下３桁）に左寄せ。
１０行目	%12.3e実数を１２桁内（小数点以下３桁）に右寄せ、 %-12.3eは実数を１２桁内（小数点以下３桁）に左寄せ。

３．１．３　キーボードから入力、画面に出力

ユーザからの入力を変数に代入する場合はSTDINを使い、結果の出力はSTDOUT、エラーの出力はSTDERRを使う。最初、STDINはキーボード、STDOUTとSTDERRは画面に設定されている。STDIN,STDOUT,STDERRは**ファイルハンドル**という。

```
1  #!/usr/local/bin/perl
2  # << p313.pl >>
3  print"好きなくだものの名前を入力してください：";
4  $fruit = <STDIN>;
5  chop($fruit);
6  print"私も$fruitが好きです。¥n";
7  exit(0);
```

実行結果

```
% perl p313.pl
好きなくだものの名前を入力してください：apple
私もappleが好きです。
```

説明

４行目	データ（数値、文字列）を保存するものとして変数がある。 データをひとつ保存する変数をスカラー変数といい、 $nameなどと書く。 スカラー変数の名前は、$、英文字を１個、（必要があれば 英文字、数字、下線を１個以上）続けたものである。 また、大文字、小文字は異なるものとして扱われる。 端末から1行文のデータを読み込むには、<STDIN>演算子を使う。 入力されたデータは、変数$fruitに代入される。 <STDIN>のみの場合,入力データはスカラー変数$_に代入される。
５行目	スカラー変数に代入された文字列データの末尾には、 改行文字が付いている。chop()関数は、文字列データの 末尾１文字を取り除く。

３．２　条件分岐

３．２．１　ｉｆ文

書き方	if(式) {第１ブロック}else{第２ブロック}
機能	式が真なら第１ブロック、偽なら第２ブロックを実行する。 {と}で囲まれた部分をブロックという。

```
 1 #!/usr/local/bin/perl
 2 # << p321.pl >>
 3 # 正整数をひとつ読み込み、偶数か奇数かを判定し出力する。
 4 print"適当な正整数を入力して下さい：";
 5 $n = <STDIN>;
 6 chop($n);
 7 if( $n%2 == 0 ) {
 8   print"正整数$nは偶数です。¥n";
 9 } else {
10   print"正整数$nは奇数です。¥n";
11 }
12 exit(0);
```

実行結果

```
% perl p321.pl
適当な正整数を入力して下さい：123
正整数123は奇数です。
```

説明

７行目	%演算子は、剰余を求める。 演算子は、１つ以上の値を基に、新しい結果を生み出す。 ==演算子は２つの数値を比較し、等しければ結果は真、 等しくなければ結果は偽となる。

数値と文字の比較演算子（$a 比較演算子 $b ）

	数値	文字列
等しい	==	eq
等しくない	!=	ne
大きい	>	gt
大きいまたは等しい	>=	ge
小さい	<	lt
小さいまたは等しい	<=	le

３．３　制御構造

３．３．１　ｗｈｉｌｅ文

書き方	**while**(式) {ブロック}
機能	式が真の間、ブロックを繰り返す。 **next**文はループの最後に飛び、次の繰り返しを実行する。 **last**文はループから脱出する。

```
 1 #!/usr/local/bin/perl
 2 # << p331.pl >>
 3 # 正整数nを読み込み、1からnまでの和を出力する。
 4 print"適当な正整数を入力して下さい：";
 5 $n = <STDIN>; chop($n);
 6 print"1から$nまでの和は";
 7 $sum = 0;
 8 while( $n > 0 ) { $sum = $sum + $n; $n = $n - 1; }
 9 print "$sumです。¥n";
10 exit(0);
```

実行結果

```
% perl p331.pl
適当な正整数を入力して下さい：10
1から10までの和は55です。
```

３．３．２　ｆｏｒ文

書き方	for(式1; 式2; 式3) {ブロック}
機能	(1)式１が設定される。 (2)式２が評価され、結果が真ならばブロックが実行される。 (3)式３が評価される。 あとは、式２が真の間、(2),(3)が繰り返される。 式２が偽になったとき終了する。 next文はループの最後に飛び、次の繰り返しを実行する。 last文はループから脱出する。

```
1  #!/usr/local/bin/perl
2  # << p332.pl >>
3  # 正整数nを読み込み、1からnまでの和を出力する。
4  print"適当な正整数を入力して下さい：";
5  $n = <STDIN>; chop($n);
6  $sum = 0;
7  for( $i=1; $i<=$n; $i=$i+1 ) { $sum = $sum + $i; }
8  print"1から$nまでの和は$sumです。¥n";
9  exit(0);
```

実行結果

```
% perl p332.pl
適当な正整数を入力して下さい：10
1から10までの和は55です。
```

３．３．３　ｆｏｒｅａｃｈ文

書き方	foreach 変数（配列）{ブロック}
機能	配列中の値を変数に代入しながら、ブロックをそれぞれの値について１回ずつ実行する。 next文はループの最後に飛び、次の繰り返しを実行する。 last文はループから脱出する。

```
1  #!/usr/local/bin/perl
2  # << p333.pl >>
3  # いくつかの正整数の和を出力する。
4  $sum = 0;
5  foreach $p ( (2,3,5,7,11,13,17,19) ) {
6    print"$p ";
7    $sum = $sum + $p;
8  }
9  print"の和は$sumです。¥n";
10 exit(0);
```

実行結果

```
% perl p333.pl
2 3 5 7 11 13 17 19 の和は77です。
```

３．４　配列

複数のデータを同じ名前で扱い、添字で個々のデータを区別することができる。このような仕組みを配列といい、配列の名前を配列名、個々のデータを配列要素という。

添字	データ
0	１２３
1	ａｂｃ
2	４５
3	６７８
4	ＸＹＺ

３．４．１　宣言、参照

```
 1 #!/usr/local/bin/perl
 2 # << p341.pl >>
 3 # 配列の宣言。
 4 @a = (123,"abc",45,678,"XYZ");
 5 # 配列の参照。
 6 print"@a¥n";
 7 print"$a[0] $a[1] $a[2] $a[3] $a[4]¥n";
 8 # 配列の要素数。
 9 print"要素数:$#a+1¥n";
10 for($i=0; $i<=$#a; $i++ ) {
11   print"$a[$i]¥n";
12 }
13 exit(0);
```

実行結果

```
% perl p341.pl
123 abc 45 678 XYZ
123 abc 45 678 XYZ
要素数：5
123
abc
45
678
XYZ
```

説明

４行目	@a = (123,"abc",45,678,"XYZ")で配列a全体に値（数値、文字列混在可）を設定する。 配列要素は、$a[0],$a[1],$a[2],$a[3],$a[4]で指定する。 すなわち、@a = (123,"abc",45,678,"XYZ")は、$a[0]=123; $a[1]="abc"; $a[2]=45; $a[3]=678; $a[4]="XYZ"と同等である。
９行目	$#配列名は、配列の最後の添字を指す。したがって、配列aの要素数は、$#a+1 で求められる。

３．４．２　代入

```
 1  #!/usr/local/bin/perl
 2  # << p342.pl >>
 3  @a = (123,"abc",45,678,"XYZ");
 4  # 代入。
 5  ($x,$y,@b) = @a;
 6  print"$x¥n";
 7  print"$y¥n";
 8  print"@b¥n";
 9  # 代入。
10  ($u,$v,$w) = @b;
11  print"$u¥n";
12  print"$v¥n";
13  print"$w¥n";
14  exit(0);
```

実行結果

```
% perl p342.pl
123
abc
45 678 XYZ
45
678
XYZ
```

説明

５行目	$x=$a[0],$y=$a[1],@b=($a[2],$a[3],$a[4])と同等。
１０行目	$u=$b[0],$v=$b[1],$w=$b[2]と同等。

３．４．３　配列の先頭に追加、削除

```
 1  #!/usr/local/bin/perl
 2  # << p343.pl >>
 3  @a = ("aaa","bbb","ccc");
 4  # 配列の先頭から削除。
 5  $x = shift(@a);
 6  print"$x¥n";
 7  print"@a¥n";
 8  # 配列の先頭に追加。
 9  unshift(@a,"AAA");
10  print"@a¥n";
11  exit(0);
```

実行結果

```
% perl p343.pl
aaa
bbb ccc
AAA bbb ccc
```

説明

５行目	配列aの先頭から要素を１つ取り除き、変数$xに代入する。
９行目	配列aの先頭に要素を追加する。

３．４．４　配列の末尾に追加、削除

```
 1  #!/usr/local/bin/perl
 2  # << p344.pl >>
 3  @a = ("aaa","bbb","ccc");
 4  # 配列の末尾を削除。
 5  $x = pop(@a);
 6  print"$x¥n";
 7  print"@a¥n";
 8  # 配列の末尾に追加。
 9  push(@a,"CCC");
10  print"@a¥n";
11  exit(0);
```

実行結果

```
% perl p344.pl
ccc
aaa bbb
aaa bbb CCC
```

説明

５行目	配列aの末尾から要素を１つ取り除き、変数$xに代入する。
９行目	配列aの末尾に要素を追加する。

３．４．５　文字列の分解と結合

```
 1  #!/usr/local/bin/perl
 2  # << p345.pl >>
 3  $s = "hcs.ipc.ibaraki.ac.jp";
 4  # 文字列の分解。
 5  @a = split(/¥./,$s);
 6  print"$a[0]¥n";
 7  print"$a[1]¥n";
 8  print"$a[2]¥n";
 9  print"$a[3]¥n";
10  print"$a[4]¥n";
11  # 文字列の結合。
12  $t = join(/¥./,@a);
13  print"$t¥n";
14  exit(0);
```

実行結果

```
% perl p345.pl
hcs
ipc
ibaraki
ac
jp
hcs.ipc.ibaraki.ac.jp
```

説明

５行目	文字列を区切り記号（.）で分割する。¥でメタキャラクタの機能を抑え、本来の文字に戻す。
１２行目	配列aの要素を区切り記号（.）でつなぐ。

３．４．６　引数を保存する配列

```
1  #!/usr/local/bin/perl
2  # << p346.pl >>
3  for($i=0; $i<=$#ARGV; $i++ ) {
4    print"ARGV[$i]=$ARGV[$i]¥n";
5  }
6  exit(0);
```

実行結果

```
% perl p346.pl a b c
ARGV[0]=a
ARGV[1]=b
ARGV[2]=c
```

説明

３行目	$#ARGVは引数配列の最後の添え字を表す。
４行目	引数は配列ARGVに保存される。

３．５　連想配列

キーとそれに対応する値の対を集めたものを連想配列という。

キー	値
red	赤
blue	青
green	緑
yellow	黄
black	黒

３．５．１　生成、代入、削除

```
 1 #!/usr/local/bin/perl
 2 # << p351.pl >>
 3 # 連想配列生成。
 4 %table = ("red"=>"赤","blue"=>"青","green"=>"緑");
 5 # 代入。
 6 $table{"yellow"} = "黄";
 7 # 削除。
 8 delete($table{"red"});
 9 if( exists($table{"blue"}) ) {
10   print"キーblueは存在します¥n";
11 }
12 exit(0);
```

実行結果

```
% perl p351.pl
キーblueは存在します
```

説明

４行目	tableは連想配列名、%tableは、連想配列全体を指す。 連想配列全体を設定するとき、(キー,値,キー,値,…)または(キー=>値,キー=>値,…)とする。 連想配列のそれぞれの要素は、キーによって参照される。 参照するときは、$table{"red"}または$table{$name}のように書く。
８行目	要素を削除するときは、delete($連想配列名{"キー"})とする。
９行目	キーの存在は、exists($連想配列名{"キー"})で確認できる。

３．５．２　連想配列のすべてのキーを表示

```
 1  #!/usr/local/bin/perl
 2  # << p352.pl >>
 3  %table = ("red"=>"赤","blue"=>"青","green"=>"緑");
 4  # すべてのキーを表示。
 5  @a = keys(%table);
 6  print"@a¥n";
 7  # 連想配列のすべての対を表示。
 8  foreach $k (keys(%table)) {
 9    $v = $table{$k};
10    print"$k $v¥n";
11  }
12  exit(0);
```

実行結果

```
% perl p352.pl
green blue red
green 緑
blue 青
red 赤
```

説明

５行目	keys関数はすべてのキーをひとつの配列にして返す。

３．５．３　連想配列のすべてのキーをアルファベット順に表示

```
 1  #!/usr/local/bin/perl
 2  # << p353.pl >>
 3  %table = ("red"=>"赤","blue"=>"青","green"=>"緑");
 4  # すべてのキーを表示。
 5  @a = keys(%table);
 6  print"@a¥n";
 7  # 連想配列のすべての対をアルファベット順に表示。
 8  foreach $k (sort(keys(%table))) {
 9    $v = $table{$k};
10    print"$k $v¥n";
11  }
12  exit(0);
```

実行結果

```
% perl p353.pl
green blue red
blue 青
green 緑
red 赤
```

説明

８行目	sort関数でキーをアルファベット順に取り出す。

３．５．４　連想配列のすべての値を表示

```
1 #!/usr/local/bin/perl
2 # << p354.pl >>
3 %table = ("red"=>"赤","blue"=>"青","green"=>"緑");
4 @v = values(%table);
5 print"@v¥n";
6 exit(0);
```

実行結果

```
% perl p354.pl
緑 青 赤
```

説明

４行目	values関数ですべての値をひとつの配列にして返す。