Ｐｅｒｌの基礎Ⅱ

０．目次

３．Ｐｅｒｌの書き方

３．６　サブルーチン

サブルーチンは、一連の文から構成され、呼び出されると実行される。

３．６．１　変数引数の処理

引数は、配列@_に入れられて関数に渡される。関数側で配列@_を変更すると呼び出し側でも変更される。

```
 1 #!/usr/local/bin/perl
 2 # << p361.pl >>
 3 $x = 123;
 4 $y = "abc";
 5 print"x=$x y=$y¥n";
 6 &func($x,$y);
 7 print"x=$x y=$y¥n";
 8 exit(0);
 9 sub func {
10   print"@_[0] @_[1]¥n";
11   @_[0] = -@_[0]; @_[1] = "ABC";
12 }
```

実行結果

```
% perl p361.pl
x=123 y=abc
123 abc
x=-123 y=ABC
```

説明

６行目	サブルーチン名の前に&を付けることで、式の中から サブルーチンを呼び出すことができる。 引数$x,$yがサブルーチンに渡される。
９行目	サブルーチンを定義するには、subのあとにサブルーチン名を 書く。
１１行目	@_は特別な配列で、サブルーチンに引き渡される値が順に 保存されている。配列@_の要素は、@_[0],@_[1],@_[2],...。 すなわち、@_[0]と$x,@_[1]と$yが同等であることを意味し、 @_[0]を変更すると$xの値も変わることになる。

３．６．２　値引数の処理

関数側で、変数の変更が呼び出し側に影響を与えない。

```
 1  #!/usr/local/bin/perl
 2  # << p362.pl >>
 3  $x = 123;
 4  $y = "abc";
 5  print"x=$x y=$y¥n";
 6  &func($x,$y);
 7  print"x=$x y=$y¥n";
 8  exit(0);
 9  sub func {
10    print"@_[0] @_[1]¥n";
11    my $a = @_[0];
12    my $b = @_[1];
13    $a = -$a; $b = "ABC";
14    print"a=$a b=$b¥n";
15  }
```

実行結果

```
% perl p362.pl
x=123 y=abc
123 abc
a=-123 b=ABC
x=123 y=abc
```

説明

１１行目	my関数は、ローカル変数aを宣言する。ローカル変数aは このサブルーチン内で有効で、呼び出し側に影響を与えない。

３．６．３　戻り値の処理

```
 1  #!/usr/local/bin/perl
 2  # << p363.pl >>
 3  $n = 10;
 4  $sum = &func($n);
 5  print"1から$nまでの和は$sumです。¥n";
 6  exit(0);
 7  sub func {
 8    my $n = @_[0];
 9    $s = 0;
10    for( $i=1; $i<=$n; $i++ ) { $s = $s + $i; }
11    return $s;
12  }
```

実行結果

```
% perl p363.pl
1から10までの和は55です。
```

説明

１１行目	return関数は、任意の位置から値と制御を呼び出し側に戻す。 return関数がないときは、サブルーチン内で最後に評価された式の値が戻り値となる。

３．６．４　変数の有効範囲

宣言された変数はファイル内で有効となる。

```
 1 #!/usr/local/bin/perl
 2 # p364.pl >>
 3 $x = 123;
 4 $y = "abc";
 5 print"x=$x y=$y¥n";
 6 &func();
 7 print"x=$x z=$z¥n";
 8 exit(0);
 9 sub func {
10   $x = 456;
11   print"x=$x y=$y¥n";
12   $z = 789;
13 }
```

実行結果

```
% perl p364.pl
x=123 y=abc
x=456 y=abc
x=456 z=789
```

３．６．５　再帰呼び出し

正整数nを読み込み、サブルーチンを用いて1からnまでの積を求め出力する。ただし、データの最後は０とする。

```
 1 #!/usr/local/bin/perl
 2 # << p365.pl >>
 3 print "適当な正整数を入力して下さい：";
 4 chop( $n = <STDIN> );
 5 $result = &fact($n);
 6 print "$n! = $result¥n";
 7 exit(0);
 8 sub fact {
 9   my $k = @_[0];
10   if( $k <= 0 ) { $w = 1; } else { $w = $k * &fact($k-1); }
11   return $w;
12 }
```

実行結果

```
% perl p365.pl
適当な正整数を入力して下さい
5! = 120
```

説明

１０行目	再帰呼び出し。

３．７　ファイルハンドル

ファイルハンドルは、ファイル、デバイス、パイプ、ソケットを統一的に扱うためのものである。

３．７．１　ファイルから入力

```
 1 #!/usr/local/bin/perl
 2 # << p371.pl >>
 3 open(FILE,"<p371.dat") || die "cannot open: $!";
 4 while( <FILE> ) {
 5   chop( $_ );
 6   ($c,$n) = split(/ /,$_);
 7   print"$c <--> $n¥n";
 8 }
 9 close(FILE);
10 exit(0);
```

ファイル：(p371.dat)

```
red 赤
blue 青
green 緑
```

実行結果

```
% perl p371.pl
red <--> 赤
blue <--> 青
green <--> 緑
```

説明

３行目	open関数はファイルハンドル（FILE）を生成し、それに ファイル（p371.dat）を結びつける。ファイルは、入力用に オープンされる。「<」は省略可能。 die関数は、"文字列"を出力し、実行を終了する。 演算子||は、左辺の演算が失敗した時右辺の演算を実行する。 変数$!はエラーの値を保存する特別な変数である。
４行目	<>は読み込みに使われる演算子で、ファイルハンドル (FILE)から１行ずつ読み込まれ、スカラー変数$_に自動的に 代入される。 ファイルの最後には、空文字が返ってくる。
９行目	ファイルハンドルを破棄する。

３．７．２　ファイルへ出力

```
 1 #!/usr/local/bin/perl
 2 # << p372.pl >>
 3 open(FILEI,"<p372.dat") || die "cannot open: $!";
 4 open(FILEO,">p372.rst");
 5 while( <FILEI> ) {
 6   chop( $_ );
 7   ($c,$n) = split(/ /,$_);
 8   print FILEO "$c <--> $n¥n";
 9 }
10 close(FILEI);
11 close(FILEO);
12 exit(0);
```

ファイル：(p372.dat)

```
red 赤
blue 青
green 緑
```

実行結果

```
% perl p372.pl
% cat p372.rst
red <--> 赤
blue <--> 青
green <--> 緑
```

説明

４行目	出力用にオープン：open(ファイルハンドル,"> ファイル名") 追加用にオープン：open(ファイルハンドル,">> ファイル名")

３．７．３　ファイルの更新

ファイル中のデータを辞書順に並べ替える。

```
 1 #!/usr/local/bin/perl
 2 # << p373.pl >>
 3 open(FILE,"+<p373.dat") || die "cannot open: $!";
 4 while( <FILE> ) {
 5   chop( $_ );
 6   ($c,$n) = split(/ /,$_);
 7   $table{$c} = $n;
 8 }
 9 seek(FILE,0,0);
10 foreach $c ( sort(keys(%table)) ) {
11   $n = $table{$c};
12   print FILE "$c $n¥n";
13 }
14 truncate(FILE,tell);
15 close(FILE);
16 exit(0);
```

ファイル：(p373.dat)

```
red 赤
blue 青
green 緑
```

実行結果

```
% perl p373.pl
% cat p373.dat
blue 青
green 緑
red 赤
```

説明

３行目	更新用にオープン：open(ファイルハンドル,"+< ファイル名")
９行目	seek(FILE,0,0)でファイルの先頭から書き込むようにする。
１４行目	tellで現在のファイルポインタの位置を取得する。 truncate(FILE,tell)で現在のファイルポインタの位置以降を削除する。

３．７．４　ファイルの属性

現ディレクトリ下のディレクトリ及びファイル表示する。

```
 1 #!/usr/local/bin/perl
 2 # << p374.pl >>
 3 $f = $ARGV[0];
 4 if( -d $f ) {
 5   print"$fはディレクトリ：";
 6   $x = -A $f;
 7   print"(最終アクセス時から経過日数：$x¥n";
 8 }
 9 if( -f $f ) {
10   print"$fは通常のファイル：";
11   $y = -M $f;
12   print"最終更新時から経過日数：$y¥n";
13 }
14 exit(0);
```

実行結果

```
% perl p374.pl p373.pl
p373.plは通常のファイル：最終更新時から経過日数：0.402835648148148
% perl p374.pl tmp
tmpはディレクトリ：最終アクセス時から経過日数：5.48881944444444
```

説明

４行目	-d $f でファイル$fがディレクトリかどうか調べる。
６行目	ファイル$fが最後にアクセスされたときからの日数を求める。
９行目	-f $f でファイル$fが通常のファイルかどうか調べる。
１１行目	ファイル$fが最後に更新されたときからの日数を求める。

ﾌｧｲﾙﾃｽﾄ演算子	意味
-d $f	ファイル$fがディレクトリなら真。
-e $f	ファイル$fが存在するなら真。
-r $f	ファイル$fが読込可能なら真。
-s $f	ファイル$fが空でないなら真。
-w $f	ファイル$fが書込可能なら真。
-x $f	ファイル$fが実行可能なら真。
-z $f	ファイル$fが空なら真。
-A $f	ファイル$fが最後にアクセスされたときからの日数
-B $f	ファイル$fがバイナリファイルなら真。
-M $f	ファイル$fが最後に更新されたときからの日数。
-T $f	ファイル$fがテキストファイルなら真。

３．７．５　ファイルロック

複数のプロセスが同時に同じファイルにアクセスしないようにする。
引数の値だけファイルに１を加えるプロセス（p375a.pl）と引数の値だけファイルから１を引くプロセス（p375b.pl）を並行に実行させる。

```
 1 #!/usr/local/bin/perl
 2 # << p375a.pl >>
 3 $n = $ARGV[0];
 4 for( $i=1; $i<=$n; $i++ ) {
 5   open(FILE,"+<p375.dat");
 6   flock(FILE,2);                ファイルをロックする。
 7   chop( $c = <FILE> );
 8   $c++;                         １を加える。
 9   seek(FILE,0,0);
10   print FILE "$c¥n";
11   truncate(FILE,tell);
12   flock(FILE,8);                ファイルのロックを解除する。
13   close(FILE);
14 }
15 exit(0);
```

```
 1 #!/usr/local/bin/perl
 2 # << p375b.pl >>
 3 $n = $ARGV[0];
 4 for( $i=1; $i<=$n; $i++ ) {
 5   open(FILE,"+<p375.dat");
 6   flock(FILE,2);                ファイルをロックする。
 7   chop( $c = <FILE> );
 8   $c--;                         １を引く。
 9   seek(FILE,0,0);
10   print FILE "$c¥n";
11   truncate(FILE,tell);
12   flock(FILE,8);                ファイルのロックを解除する。
13   close(FILE);
14 }
15 exit(0);
```

６行目	ファイルをロックする。
１２行目	ファイルのロックを解除する。

実行結果

```
% cat p375.dat
0
% perl p375a.pl 1001 &; perl p375b.pl 1000 &
増加プロセス：10 和:10
増加プロセス：20 和:20
増加プロセス：30 和:30
増加プロセス：40 和:40
[1] 11399
増加プロセス：50 和:50
＜＜途中省略＞＞
増加プロセス：190 和:190
[2] 11400
減少プロセス：10 和:182
減少プロセス：20 和:172
減少プロセス：30 和:162
減少プロセス：40 和:152
減少プロセス：50 和:142
減少プロセス：60 和:132
増加プロセス：200 和:138
増加プロセス：210 和:148
増加プロセス：220 和:158
増加プロセス：230 和:168
増加プロセス：240 和:178
＜＜途中省略＞＞
減少プロセス：960 和:23
増加プロセス：990 和:24
減少プロセス：970 和:23
増加プロセス：1000 和:24
減少プロセス：980 和:21
減少プロセス：990 和:11
減少プロセス：1000 和:1

[2]    Done                    perl p375b.pl 1000
[1]  + Done                    perl p375a.pl 1001
% cat p375.dat
1
```

３．７．６　メールの送信

```
 1 #!/usr/local/bin/perl
 2 # << p376.pl >>
 3 while( true ) {
 4   print"メールの宛先をどうぞ¥n";
 5   $mail_address = <STDIN>;
 6   chop($mail_address);
 7   if( $mail_address eq "" ) { last; }
 8   open(MAIL,"|mailx $mail_address");
 9   print MAIL "Hello! ¥n";
10   close(MAIL);
11 }
12 exit(0);
```

実行結果

```
% perl p376.pl
メールの宛先をどうぞ
xxx@hcs.ipc.ibaraki.ac.jp
```

説明

８行目	open(ファイルハンドル,"|コマンド");は、ファイルハンドルを介してコマンドに出力を渡す。
９行目	出力は、ファイルハンドル（MAIL）に送られる。すなわち、出力されたメッセージは、mailxコマンドの入力になる。

３．７．７　時刻の表示

```
1 #!/usr/local/bin/perl
2 # << p377.pl >>
3 open(TIME,"date |");
4 $now = <TIME>;
5 chop($now);
6 close(TIME);
7 print"現在時刻:$now¥n";
8 exit(0);
```

実行結果

```
% perl p377.pl
現在時刻:2005年03月05日 （土） 12時56分32秒 JST
```

説明

３行目	open(ファイルハンドル,"コマンド|");は、ファイルハンドルを介してコマンドからの出力を受け取る。

３．８　UNIXコマンド実行

３．８．１　コマンドの実行と表示

system関数は、コマンドを実行して結果を画面に表示する。

```
1  #!/usr/local/bin/perl
2  # << p381.pl >>
3  system("date");
4  system("sleep 5");
5  system("date");
6  exit(0);
```

実行結果

```
% perl p381.pl
2005年03月05日 （土） 13時03分48秒 JST
2005年03月05日 （土） 13時03分53秒 JST
```

説明

３行目	コマンドdateを実行する。出力先は画面。
４行目	５秒間スリープ。

３．８．２　コマンドの実行と結果の取得

```
1  #!/usr/local/bin/perl
2  # << p382.pl >>
3  $x = `date`;
4  chop( $x );
5  print"$x¥n";
6  exit(0);
```

実行結果

```
% perl p382.pl
2005年03月05日 （土） 13時24分38秒 JST
```

説明

３行目	コマンドdateを実行する。結果は変数$xに代入される。

３．９　ｐｅｒｌプログラムの再利用

require関数を使うと、必要なときにperlプログラムを読み込むことができる。perlプログラムは特殊変数@INCに指定されたディレクトリから探される。中に、カレントディレクトリが含まれているので、perlプログラムは、読み込むプログラムと同じディレクトリにおいておけばよい。

require関数で読み込まれたperlプログラム は、即実行される。その結果(最後に評価した値)が真(１)でないと停止する。そのため、perlプログラムの最後に、1; を書いておく。

３．９．１　require関数、特殊変数@INCのチェック

```
1  #!/usr/local/bin/perl
2  # << p391.pl >>
3  require "p391a.pl";
4  # 特殊変数@INCの表示。
5  foreach $x ( @INC ) { print"$x¥n"; }
6  # perlプログラムに含まれているサブルーチン（test1,test2）に実行。
7  &test1();
8  &test2();
9  exit(0);
```

```
1  # << p391a.pl >>
2  sub test1 {
3    print"-----¥ntest1¥n-----¥n";
4    return;
5  }
6  sub test2 {
7    print"-----¥ntest2¥n-----¥n";
8    return;
9  }
10 1;
```

実行結果

```
% perl p391.pl
/usr/local/lib/perl5/5.8.3/sun4-solaris
/usr/local/lib/perl5/5.8.3
/usr/local/lib/perl5/site_perl/5.8.3/sun4-solaris
/usr/local/lib/perl5/site_perl/5.8.3
/usr/local/lib/perl5/site_perl
.
-----
test1
-----
-----
test2
-----
```

３．９．２　１行分のデータを空白で区切り単語に分割

```
 1 #!/usr/local/bin/perl
 2 # << p392.pl >>
 3 require "p392a.pl";
 4 while( $line = <STDIN> ) {
 5   chop( $line );
 6   if( $line eq "" ) { last; }
 7   @a = &linetodata($line);
 8   print"main: $line¥n";
 9   foreach $x ( @a ) {
10     print"$x¥n";
11   }
12 }
13 exit(0);
```

```
 1 # << p392a.pl >>
 2 sub linetodata {
 3   my $x = $_[0];
 4   @b = split(/ +/,$x);
 5   return(@b);
 6 }
 7 1;
```

実行結果

```
% perl p392.pl
11 aaa   bbb  789
main: 11 aaa   bbb  789
11
aaa
bbb
789

%
```