Perl 入门和提高　　Lesson 4

周晓方

Statements --- *perlsyn*

- Perl语法：查看*perlsyn*网页
- 复合语句，或叫块语句 {语句;语句;…语句;}
- Comments in Perl

```
# This is the perl style comments.
/* C style */ is not for perl, nor // C++ sytle comments
```

- Comment out block of code

```
if (0) {
…
… some valid perl code
}
```

- PoC (Plain old comments)

```
# line 200 "bzzzt"
# the previous '#' must on the first column
die 'foo'; #格式是 顶格的'#' 可选的空格 行号 "文件名"
```

```
foo at bzzzt line 201.
```

其他语言中PoC的例子

- 以下.y文件(片段)和对应的.c文件(片段)

```
…
242     m_list:
243                 m_list m_level_declaration
244                 {
245                         $$ = [@{$1}, $2];
246                 }
247     |           m_level_declaration
…
```

```
…
4318    case 2:
4319    #line 244 “foobla.y"
4320    {
4321            yyval = [@{yyvsp[-1]}, yyvsp[0]];
4322    }
4323    break;
4324    case 3:
4325    #line 248 “foobla.y"
…
```

倒装修饰的简单语句

- Simple statement + modifier

```
Simple_stat modifier Cond;
#if/unless/while/until/foreach
 !!Always evaluates Condition before execute Expression!!

 do BLOCK while cond; run BLOCK once before evaluate cond
     print " \$A is negitive!" if $A < 0;
     die "SOS!" if ($fail);
     $B=1/$A unless $A == 0;
     ($sum, $j) = (0, 1); do { $sum += $j } while ++$j <= 100;
     $sum = 0; $sum += $_ foreach (1..100);  # loops on $_

 $hash{$key} = $v unless defined($hash{$key}); # avoid over-write

 $I=1; $J=0; $J+=$I++ until ($I > 10); print $J; # ==> 55

 perl -e "print while <>;" < readme.txt
```

Control Flow 1

- False conditions:

```
0.000, 0, undef, "0", ""
(but "0.0" "00" is true)
```

- If statement:

```
if (…) {
   …
}
```

```
if (…) {
   …
} else {
   …
}
```

```
# compare modified statement

expression if …;
expression unless …;
```

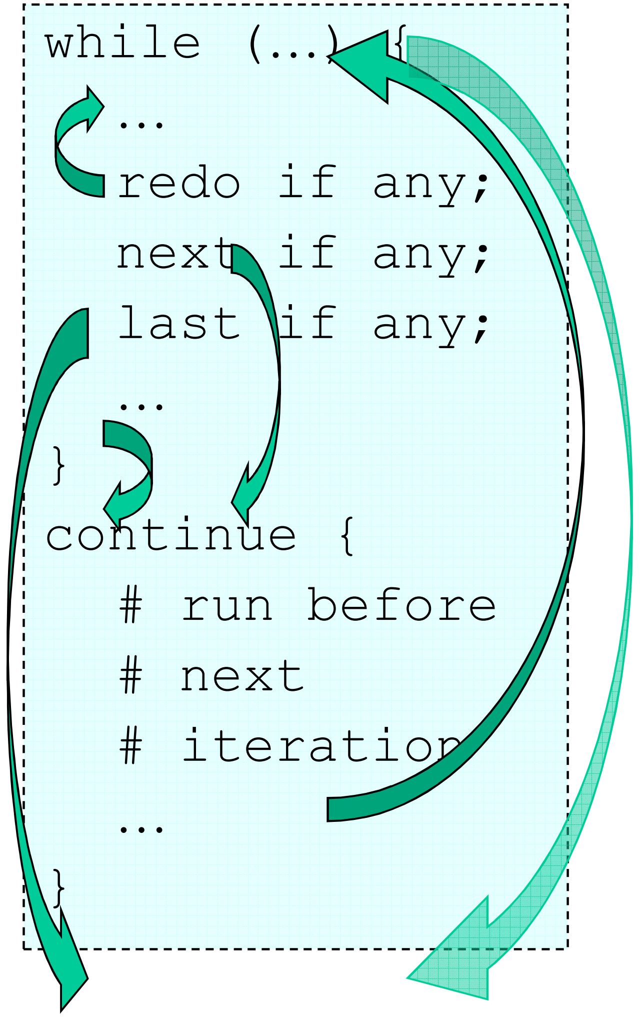
```
# Multi-choice:
# not "else if"
# not "elseif"
# but "elsif"

if (…) {
   …
} elsif (…) {
   …
} elsif (…) {
   …
} elsif (…) {
   …
} else {
   …
}
```

Control Flow 2

- *while* loop — *until* loop
  - next ➔jump to continue block and check condition
  - redo ➔jump to beginning of loop, no continue block or cond check
  - last ➔exit loop immediately

```
# check
# before loop
while (…) {
   …
}
```

```
# loop before
# check
do {
   …
} while (…);
# redo, next,
  last not
  allowed.
```



```
while (…) {
   …
   redo if any;
   next if any;
   last if any;
   …
}
continue {
   # run before
   # next
   # iteration
   …
}
```

```
# check
# before loop
until (…) {
   …
}
```

```
# loop before
# check
do {
   …
} until (…);
# similar to
  do{} while
```

Control Flow 3

- *for* loop

```
for (start_exp; condition_exp; step_exp) {
  …
}
```

```
for ($n=1, $sum=0; $sum<=1000; $n++) {
  $sum += $n;
}
print "n=$n; sum=$sum\n";
$sum -= $n--;
print "n=$n; sum=$sum\n";

# n=46; sum=1035
# n=45; sum=989
```

Control Flow 4

- ***foreach*** loop

```
my $var; …
foreach $var (@list) {
   …don't splice the @list here…
} # $var is local to the loop
```

```
# ! Side effect of foreach ! foreach循环有副作用
@array = (1..5, 5..10);
print("@array\n");
foreach (@array) {       # Say 'for (@L)…' is also ok
    $_ = "Five" if ($_ == 5); #default loop var is $_
} # foreach loop is faster than for loop
print("@array\n");

# 1 2 3 4 5 5 6 7 8 9 10
# 1 2 3 4 Five Five 6 7 8 9 10
```

```
foreach $var (0..10) { $var *= $var; } #non lvalue
```

Control Flow 5

- Jump keywords: *last, next, redo, goto*
  - Avoid *goto*, always write the "goto-less programs"
- Label of loops (optional, but sometimes useful)

```
OUTERLOOP:
foreach $a (@list) {
   INNERLOOP: while ($b) {
        next INNERLOOP if $c;
        next OUTERLOOP if $d;
   }
}
```

- Switch
  (see *perlsyn* for more)

```
SWITCH: {
        if (/^abc/) { $abc = 1; last SWITCH; }
        if (/^def/) { $def = 1; last SWITCH; }
        if (/^xyz/) { $xyz = 1; last SWITCH; }
        $nothing = 1;
    }
```

perlpod—Plain old document

- Mixed perl code and perl document.
- Begin with Lines '=*pod_cmd pod_parameter*'
- End with **'=cut'**，i.e. return to perl code.
- Some pod command，前后都加一个空行
  - = head1 *Your head line here*
  - = head2 *Your head line here*
  - = over *optional_indent_width*   项目列表开始
  - = item *， 或者连续的数字，或者其他字符串
  - = back   项目列表结束
- pod过滤命令：pod2text 或者 pod2html
- 具体实例，参考086pod.pl以及*perlpod*等

~~作业: 剪贴板监视程序~~(已过时)

- 写一个程序(Win32环境)，始终监视并显示剪贴板的变化，当剪贴板出现"https://"开头的文本字符串~~时，退出。附件名 *学号04.pl*~~
- 提示:看Win32::Clipboard帮助文件
- 下面是某次运行过程显示的结果

```
Clipboard changed
 text : "note that you "
Clipboard changed
 text : "#!/usr/bin/perl -w
use str"
Clipboard changed
 not text.
Clipboard changed
 text : "https://courses.xfzhou.homeftp.org/"
```

Perl模块功能强大，且大多采用面向对象的写法，大家可以通过尝试这个剪贴板模块，逐步熟悉perlobj的用法。下面是*$实例->方法(参数)* 的一些例子：

调用模块

```
use Win32::Clipboard;
```

生成一个剪贴板对象

```
my $Clip =
        Win32::Clipboard();
```

监视剪贴板变化

```
$Clip->WaitForChange()
```

判断是否是字符格式

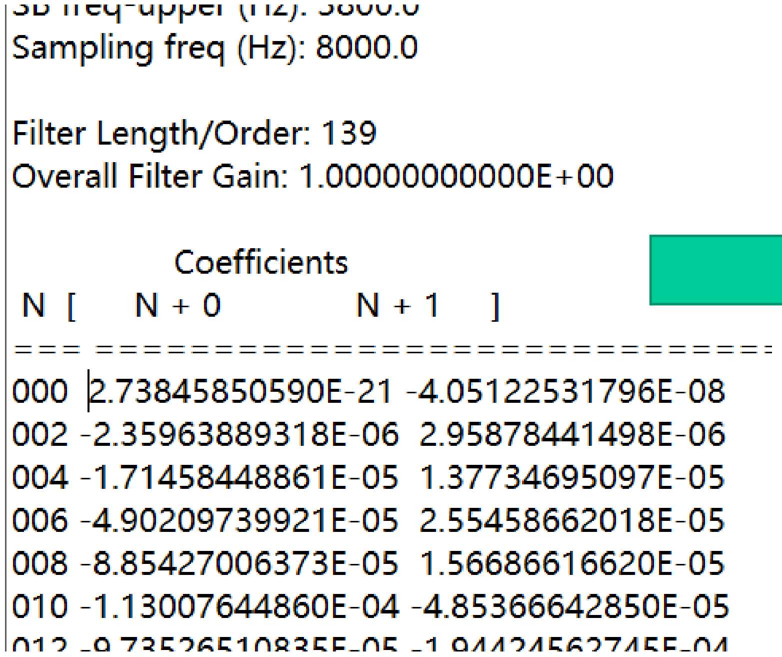
```
$Clip->IsText()
```

获得剪贴板的文本内容

```
$Clip->GetText()
```

作业04: 显示FIR冲击响应曲线

- 用列表或标量环境的<>从标准输入抓取样例文本中的系数序列，用Chart::Lines模块显示曲线并保存为png格式图片文件。生成的图片名称为学号-04.png，程序附件名 *学号-04.pl*
- 例如 17300450678-04.pl < filter.txt 保存到17300450678.png
- 提示：检测"===",开始读数据，最后行可能只有一个系数



```
SB freq-upper (Hz): 3600.0
Sampling freq (Hz): 8000.0

Filter Length/Order: 139
Overall Filter Gain: 1.00000000000E+00

            Coefficients
 N [     N + 0          N + 1     ]
=== ================================
000 2.73845850590E-21 -4.05122531796E-08
002 -2.35963889318E-06  2.95878441498E-06
004 -1.71458448861E-05  1.37734695097E-05
006 -4.90209739921E-05  2.55458662018E-05
008 -8.85427006373E-05  1.56686616620E-05
010 -1.13007644860E-04 -4.85366642850E-05
012 -9.73526510835E-05 -1.94424562745E-04
```

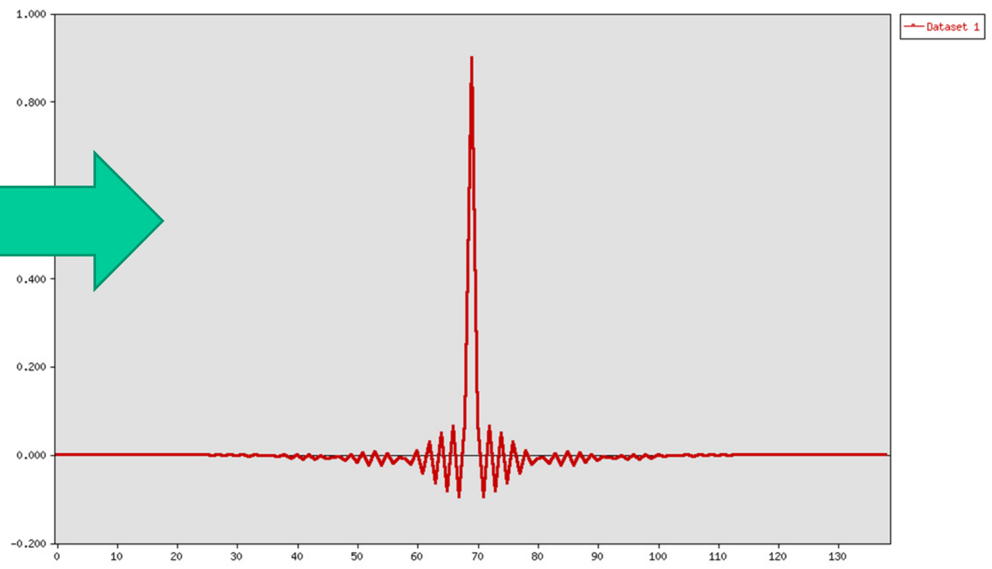
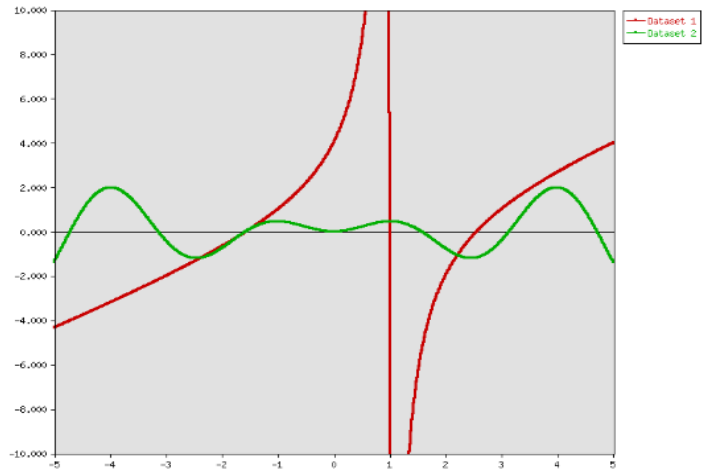


Chart::Lines示例



- CPAN / *install Chart::Lines*会安装*GD*和*Chart*下所有模块

```
#!/usr/bin/perl -w
use strict;
use Chart::Lines;
my @x = map $_ / 100, -500..500;
my @y1 = map {$_ == 1 ? 0 : ($_*$_-$_- 4)/($_-1)}  @x;
my @y2 = map {$_ * sin($_) * cos($_)} @x;
my $chart = Chart::Lines->new(800, 600);
$chart->add_dataset(@x);
$chart->add_dataset(@y1);
$chart->add_dataset(@y2);
$chart->set('skip_x_ticks' => 100,
       'max_val' => 10,'min_val' => -10);
$chart->png('temp.png');
1;
```

Subroutine I

- 参数在@_中，直接修改@_数组的元素 $_[*i*] 有副作用
- 子程序名称避免全大写
- Declare all local variables using *my(…)*
- 如果预先申明的函数, 则在调用时可以省略括号
  - sub 函数名;
  - use subs qw(函数名 函数名 函数名…);
- Check context with *wantarray( )*确定调用的上下文
- 调用时,函数名可加前缀&(强烈不推荐), 其中&foo相当于 foo(@_)

```
sub name;                    # pre declaire, 一般没有必要预先声明
                             # 但先作函数申明或定义,调用函数时就可以省略括号
$res = name($a, $b);         # call subroutine
@res = name $a, $b;          # also can say &name($a, $b)

sub name {                   # define subroutine
    my($arg1) = shift;       # copy arguments
    my($arg2) = shift;
    …
    my($result, @result);# declare local variables
    …
    …
    wantarray() ? @result : $result;  # return result
}
```

Subroutine II

- Copy arguments from @_ (子程序先复制@_的内容)

```
sub name {                # define subroutine
    my $arg1 = shift @_;
    my $ary2 = shift;       # the same as shift @_
    my($arg3, $arg4) = @_; # copy arguments
    my(@list) = @_;           # or in this format
    …
}
```

- Function with side effect, @_传递实际参数的别名 (直接修改@_的元素,有副作用)

```
sub Side_effect {
   $_[0] = $_[0] * 2;
}
my $a = 5;
Side_effect($a); # $a becomes 10 now.
Side_effect(5);  # fatal run-time error. 5 is constant
```

- 对@_作shift，不改变数组的元素，无副作用

算pi的一个例子程序

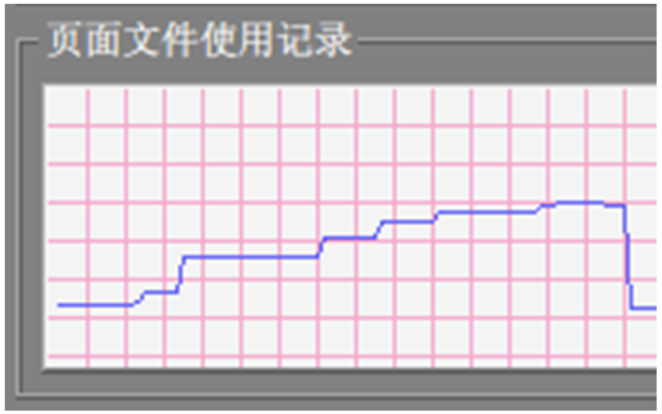
- 蒙特卡洛单位圆法近似计算π

```
#!/usr/bin/perl -w
use strict;

print "10:\t" , pi(10), "\n";
print "100:\t" , pi(100), "\n";
print "1000:\t" , pi(1000), "\n";
print "10000:\t" , pi(10000), "\n";
print "100000:\t" , pi(100000), "\n";
print "1000000:\t" , pi(1000000), "\n";

sub pi {
  my($count) = $_[0];
  my($inside);

  $inside +=  sqr(rand 1) + sqr(rand 1) < 1
     while $count-- > 0;
  4 / $_[0] * $inside;
}

sub sqr {
  my($n) = shift;
  return $n * $n;
}

1;
```

算pi的又一个例子程序

```
#!/usr/bin/perl -w
use strict;

print "10:\t" , pi(10), "\n";
print "100:\t" , pi(100), "\n";
print "1000:\t" , pi(1000), "\n";
print "10000:\t" , pi(10000), "\n";
print "100000:\t" , pi(100000), "\n";
print "1000000:\t" , pi(1000000), "\n";

sub pi {
   my($count) = shift;
   my(@distance) =
        map sqr(rand(1)) + sqr(rand(1)) < 1, 1..$count;
   4 / $count * scalar grep /1/, @distance;
}

sub sqr {
   my($n) = shift;
   return $n * $n;
}

1;
```

- 算法相同
- 占用更多资源



"*my*", "*local*", and "*our*"(perl5.6+),"state"(5.10+)

- 字典作用域 *my* declares a lexical variable totally hidden from the outside world
  - 只用于当前作用域，不自动传递给所调用的子程序
  - 可以模仿C语言的auto变量
  - 要模仿C语言的static变量，可以这样写

```
{ my($s) = 0;    # 单独用一对{}，在里面定义my变量和子程序
  sub getnext { $s++; }
} # $s对外不可见，但每次调用getnext之间都保留$s的值
print getnext; print getnext; print getnext;
```

- 动态作用域 *local*
  - 自动传递给所调用的子程序
- 全局作用域 *our* (perl5.6+)
  - use vars qw(变量列表),例`use vars qw($frob @mung %seen);`
  - 缺省情况下都是全局变量,可以用$v, $::v, 或$main::v引用
  5.10+ 局部静态变量， state， use feature 'state';

Sample

```
a();
our $x=7; our $y = 17;
a();
sub a {
  my $x = 10; local $y = 5;
  print "ax$x, ay$y\n";
  b();
}
sub b {
  print "bx$x, by$y\n";
}
```

ax10, ay5 有字典变量x，有动态变量y，都是在a中定义的
bx, by5 没有全局、动态和字典变量x，有动态变量y(在a中定义的)
ax10, ay5 有字典变量x，有动态变量y，都是在a中定义的
bx7, by5 有全局变量x(用our定义的)，有动态变量y(在a中定义的)