Perl 入门和提高　　Lesson 7

周晓方
courses@xfzhou.homeftp.org

Perl Ref (引用) 1--- basic

- Reference: \

```
$Rscalar=\$scalar;
$Rarray =\@array;
$Rhash = \%hash;
$Rfunc = \&func;
$Rref  = \$Rscalar;
```

- Dereference: **% @ % &**

`${$Rscalar}` is a scalar

`@{$Rarray}` is an array

`%{$Rhash}` is a hash

`&{$Rfunc}` is a func call

`${${$Rref}}` is a scalar

The '{' '}'s are optional

- Refer to anonymous array, hash, and function

```
$Ra = [ 'A', 55, 1..6, 'oops' ];
$Rh = { 'dog'=>'bark', 'cat'=>'mew', };
$Rf = sub {my $x = shift; return 1/$x;};
print "@$Ra\t$$Ra[7]\tDog=>$$Rh{dog}\n";
print &$Rf(5), "\t$Ra\t$Rh\t$Rf\n";
```

Perl Ref 2--- example

- A more stranger way to generate / access the reference to an array, or a hash

```
@$Rs = ('A', 55, 'oops');   #unusual
$Rt = \@arrB; @$Rt = ('B', 66); # @arrB changed
push @$list, @append;  #very useful
```

- Example: Passing 2 arrays to a function

```
@res = vec_add([1..10], [21..30]);
  ……
sub vec_add {      # illustration only. Not in good style.
    my($arr1, $arr2) = @_;

    my(@arr1) = @{$arr1};
    my(@arr2) = @{$arr2};
    my(@result);
    for (my $n=0; $n<@arr1; $n++) {
            $result[$n]=$arr1[$n]+$arr2[$n];
    }
    return @result;
}
```

Perl Ref 3--- the *ref* ( ) function

- ref() function return type of reference

```
$NotRef = 123;          ==> undef
$Rscalar=\$scalar;      ==> "SCALAR"
$Rarray =\@array;       ==> "ARRAY"
$Rhash = \%hash;        ==> "HASH"
$Rfunc = \&func;        ==> "CODE"
$Rref  = \$Rscalar;     ==> "REF"
$Robj = $some_perl_obj ➔ Object name
```

Improvements to the code on previous page:

```
4       my($arr1, $arr2) = @_;
5       return () if ref($arr1) ne "ARRAY" or ref($arr2) ne "ARRAY";
6       my(@arr1) = @{$arr1}; … …
```

Perl Ref 4 -- Ref to list/hash

- Short hands(注意优先级，perl的$$a[j]和C语言的*x[j]不同)

  ${$a} ≡ $$a, @{$a} ≡ @$a, %{$a} ≡ %$a, etc

- Ref to list: $a = `[1, 2, 3, "AAB"]`

  `${$a}[0]=50; $$a[0]=50; $a->[0]=50;` # same

- Ref to hash:$h= {cat=>"rat", dog=>"meat"}

  `${$h}{rat}="rice"; $$h{rat}="meat"; print $h->{rat};`

- Create 2D array (List of List / Array of Array) using ref:

  (-> between `[]` `{}` can be ommited)

```
$c = [{name=>"Zhou", teach=>["perl","c","soc"]},
      {name=>"Li", teach=>["VHDL", "verilog", "layout"]}
];
print "$c->[0]->{name}\n"; #same as $c->[0]{name}
print "@{$c->[1]{teach}}"; #@$c->[1]{teach}'s wrong,注意优先级
```

Perlref 5– assign v.s. push

- Create a list using assign or push:

```
my(@ref3D, $r3D);
$ref3D[3]->[2][1]=5; $ref3D[0][0][0]=1;
# same as:
@ref3D = ([[1]], undef, undef,
 [undef, undef, [undef, 5]]);
#############
$r3D->[3][2][1]=5; $r3D->[0][0][0]=1;
#same as:
$r3D = [[[1]], undef, undef,
 [undef, undef, [undef, 5]]];
#############
my $arr = [[1, 0, 0], [0, 1, 0]];
print $arr->[0][1];
push @$arr, [0, 0, 1];
```

Perlref 6– always '*use strict;*'

- '*perl -w*' and '*use strict;*' can be a big help when using perlref!
- Perl will generate new variables on the fly without any warning, but '*perl -w*' and '*use strict;*' can stop that.

```
my $aref = [
  [ "fred", "barney", ],
  [ "homer", "bart", "marge", ],
  [ "george", "jane", "elroy", ],
];
print $aref[2][2];   # Error!
  # There's no variable called '@aref'
print $aref->[2][2]; # Correct!
```

Perlref 7– common mistakes

- Generate a list like this:

  ([…], […], […])

```
for $i (0..9) {
   @arr = somefunc($i);
   $AoA[$i] = @arr;# WRONG!
} # 语法问题，标量=数组
```

```
for $i (0..9) {
   @arr = somefunc($i);
   @{$AoA[$i]} = @arr;
# Bad! If $AoA[$i] has
# assigned with \@other_arr,
# The above line may
# overwrite the @other_arr
} # 副作用，可能覆盖已有数组
```

```
for $i (0..9) {
   @arr = somefunc($i);
   $AoA[$i] = \@arr;#WRONG!
} # 语义问题，反复引用同一个数组
```

```
for $i (0..9) {
   my @arr = somefunc($i);
   $AoA[$i] = \@arr; # ?!?
} # correct but too tricky
```

```
for $i (0..9) {
   @arr = somefunc($i);
   $AoA[$i] = [@arr]; # OK!
} # 引用数组的匿名拷贝,正确并推荐!
```

```
push @AoA, [somefunc($i)] foreach 0..9; #推荐,当@AoA为空
```

- Always constructs a ref to arr/hash with […], {…}

Perlref 8 -- Garbage collection

- Perl对引用计数，遇到}减计数，归0删除

```
my($V1);
if ($true) {
   my($V2) = [1, 2, 3, 4];
   $V1 = $V2;
}
print "@$V1\n";
```

- Bad code and correction: 避免循环引用

```
if ($true) {
   my $a = 5;        # counter for $a is set to 1
   $a = \$a;         # counter for $a is 2 now
...               # counter for $a is 1, not 0,
...               # but no way to release its memory.
}                 # A cause of memory leakage.
```

这里插入 `$a = undef;` 在离开作用域前打断循环引用

Perlref 9-- Data::Dumper

- Let's usage Perl module! Dump complext structure:

```
use Data::Dumper;
my(@ref3D, $r3D);
$ref3D[3][2][1]=5;
$ref3D[0][0][0]=1;
$r3D->[3][2][1]=5;
$r3D->[0][0][0]=1;
print Dumper(@ref3D);
print Dumper($r3D);
```

```
$VAR1 = [
          [
            1
          ]
        ];
$VAR2 = undef;
$VAR3 = undef;
$VAR4 = [
          undef,
          ${\$VAR4->[0]},
          [
            ${\$VAR4->[0]},
            5
          ]
        ];
$VAR1 = [
          [
            [
              1
            ]
          ],
          undef,
          ${\$VAR1->[1]},
          [
            ${\$VAR1->[1]},
            ${\$VAR1->[1]},
            [
              ${\$VAR1->[1]},
              5
            ]
          ]
        ];
```

Perlref 10--Coderefs

- reference to subroutine or anonymous block

```
my $rfA = \&one_sub;
my $rfB = sub {print "@_\n";};
```

- Calling the code reference

```
&$rfA(1, l2); &$rfB('A', 'World');
$rfA->(1, 12); $rfB->('A', 'World');
```

- More on powerful grep( ) and map( )

```
@b = grep { $_ > 5 } @a;
map {BLOCK} list
map {+EXPR, ... } list
sort {BLOCK} list
```

Perlref 11—typeglob 类型通配

- 全局标量(不是my标量)可以使用typeglob
- 需要屏蔽标量检查 no strict 'vars';
- *new = *old 使得所有名称为new的标量、数组、散列、句柄、格式成为old的别名
- *new = \$old 使得$new成为$old的别名，而@new、%new等不变
- @v是一个数组, $Aref = *v{ARRAY} 使得$Aref成为@v引用的别名
- 传递文件句柄是会用到typeglob

```
$fout = *STDOUT; $fin = \*STDIN;
```

- 尽量避免使用typeglob，详细手册见perldata

Perlref 12—Attentions

- 带\的list并不是list的引用

  `\($a, $b, 'AA')`相当于`(\$a, \$b, \'AA')`

  `\(@A)`相当于`(\$A[0], \$A[1], \$A[2]…)`
- 都写作{}, 区分Hash的引用和BLOCK
  - 理解为BLOCK `sub f { {; @_} }`
  - 理解为HASH `sub f { +{@_} }`
  - 理解为HASH `sub f {return {@_}}`
  - 理解为BLOCK `sub f { {@_} }`
- 区分

```
$cubic[1][2][3] = 4;    #顶层是@cubic
$cubic->[1][2][3] = 4; #顶层是$cubic
$ref->[0] = 1;          # $ref是数组的引用
$ref->{0} = 1;          # $ref是Hash的引用
$ref->(0, 1);           # $ref是代码的引用
```

Perlref 13—more readings

- Symbolic ref (not covered here), see perlref

- perlreftut —Short tutorial to perl reference
- perlref —Details of perl ref.
- perllol —Array of array in perl
- perldsc—Perl data structure cookbook, LoL, LoH, HoL, HoH, and many more complex structures.
- perldata —Typeglobs, reference to file handles

- perlobj —Turn a hash-ref to perl object,
后续课程会讲解Perl模块和面向对象的Perl

GraphViz节点和连线图

- Graphviz 开源，来自AT&T/Bell Labs Innovation
  - https://graphviz.org/download/ 安装各自OS的对应版本
- perl模块GraphViz(旧)和GraphViz2(新)
- 构造对象my $g = GraphViz->new();

  width/height =>英寸, layout=>dot/neato/twopi/circo/fdp

  directed=>1/0, bgcolor=>"*h,s,v*"/"green"/etc…
- 添加节点$g->add_node('name', label=>'string');

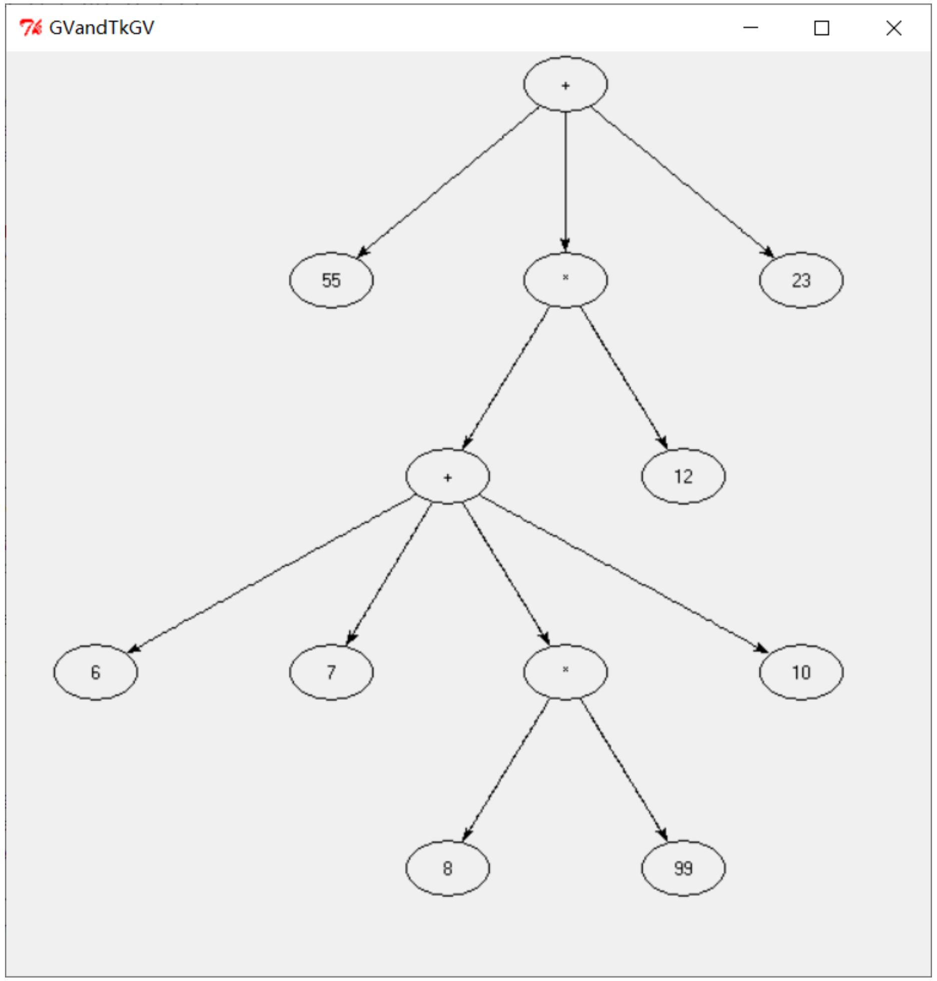
  shape=>'record/plaintext/ellipse/circle/egg/triangle/box/diamond/trapezium/parallelogram/house/hexagon/octagon'
- 添加连线$g->add_edge('name1'=>'name2');

  label=>'string', dir=>'forward/back/both/none'

  from_port/to_port=>数字
- 输出图片print $g->as-png; as_canon, as_jpeg…

联合Tk::GraphViz显示

- 安装模块Tk，Tk::GraphViz
- 新窗体:my $m = new MainWindow;
- Gv画板:my $gv =
  $m->GraphViz(
    -width => 600,
    -height => 600
    #长宽单位是像素
) -> pack;
- 插入GraphViz对象:
  $gv->show($g);
- 消息循环:MainLoop();
- 实例见
  076GVandTkGV.pl



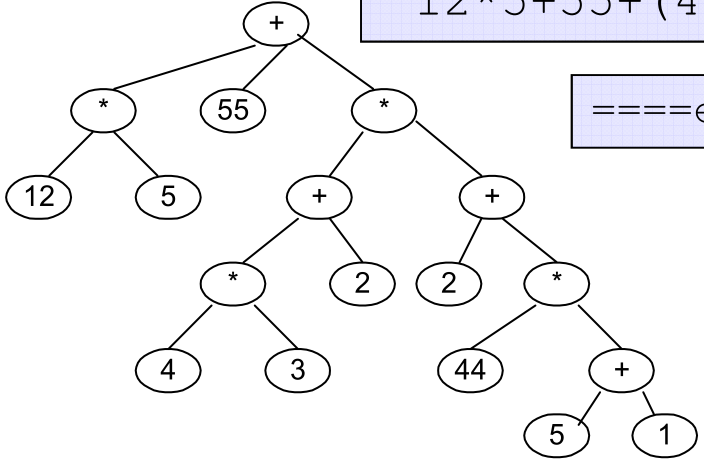
Homework

- 每次读入一个整数表达式(只包含加法、乘法、括号和整数，可含空格，不考虑单目加)。先将表达式转化成树，树用递归方式表示，每个节点表示成[op, node1, node2, node3…]，op可以是+、*，node可以是整数或另一个节点。用Data::Dumper打印树，用GraphViz结合Tk::GraphViz弹出窗体画出多叉树，最后历遍树求表达式的值。
- *学号-07.pl*

```
12*5+55+(4*3+2)*(2+44*(5+1))
```

```
====evaluate result: 3839
```

```
[
  '+',
  [
    '*',
    '12',
    '5'
  ],
  '55',
  [
    '*',
    [
      '+',
      [
        '*',
        '4',
        '3'
      ],
      '2'
    ],
    [
      '+',
      '2',
      [
        '*',
        '44',
        [
          '+',
          '5',
          '1'
        ]
      ]
    ]
  ]
];
```



+
*
55
*
12
5
+
+
*
2
2
*
4
3
44
+
5
1