第二部分
高级特性

第9学时　其他函数和运算符

Perl遵循的传统原则是“一件事情可以使用许多方法来完成”。在本学时中，我们将要更加深入地掌握这个原则。我们将要学习丰富多彩的新函数和运算符。

为了进行标量搜索和操作，到现在为止我们一直使用正则表达式。不过我们可以使用多种方法来完成这项任务，Perl提供了各种各样的函数，以便对标量进行搜索和编辑。在本学时中，我们将要介绍其他的几种方法。

另外，我们介绍了作为项目的线性列表的数组，你可以使用 foreach迭代通过这些列表，或者使用join将它们组合起来，构成标量。在本学时中，我们将要介绍一种观察数组的全新方法。

最后，我们要重新介绍一下常用的 print函数，并且给它增加一点特性。使用新的改进后的print函数，你就能够编写格式优美、适合向他人展示的报表。

在本学时中，你将要学习：

- 如何对标量进行简单的字符串搜索。
- 如何进行字符替换。
- 如何使用print函数。
- 如何将数组用作堆栈和队列。

9.1　搜索标量

正则表达式非常适合对标量进行搜索，以便找出你要的模式，但是有时使用正则表达式来搜索标量有点像杀鸡用牛刀的味道。在perl中，对模式进行组装，然后在标量中搜索该模式，需要花费一定的开销，不过这个开销并不大。另外，当你编写正则表达式时，很容易出错。为此，perl提供了若干个函数，用于对标量进行搜索，或者从标量中取出简单的信息。

9.1.1　用index进行搜索

如果你只想在另一个标量中搜索单个字符串，Pert提供了index函数。index函数的句法如下：

```
index string, substring
index string, substring, start_position
```

index函数从string的左边开始运行，并搜索 substring。index返回找到substring时所在的位置，o是指最左边的字符。如果没有找到 substring，index便返回-1。被搜索的字符串可以是字符串直接量，可以是标量，也可以是能够返回字符串值的任何表达式。 substring不是一个正则表达式，它只是另一个标量。

请记住，你编写的Pert函数和运算符可以带有包含参数的括号，也可以不带。下面是一些例子：

```
index "Ring around the rosy", "around";     # Returns 5
index("Pocket full of posies", "ket");      # Returns 3
$a="Ashes, ashes, we all fall down";
index($a, "she");                           # Returns 1
index $a, "they";                           # Returns -1 (not found)
@a=qw(oats      peas      beans);
index join(" ", @a), "peas";                # Returns 5
```

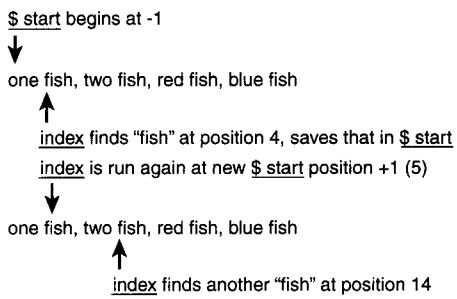
根据情况，可以给index函数规定一个字符串中开始进行搜索的起始位置，如下面的例子显示的那样。若要从左边开始搜索，使用的起始位置是 0：

```
$reindeer="dasher dancer prancer vixen";
index($reindeer, "da");          # Returns 0
index($reindeer, "da", 1);       # Returns 7
```

也可以使用带有起始位置的 index函数，以便“ 遍历 ”一个字符串，找到出现一个较短字符串的所有位置，如下所示：

```
$source="One fish, two fish, red fish, blue fish.";
$start=-1;
# Use an increasing beginning index, $start, to find all fish
while(($start=index($source, "fish", $start)) != -1) {
    print "Found a fish at $start\n";
    $start++;
}
```

上面这个代码滑动通过 $ source，如下所示：



9.1.2 用rindex向后搜索

函数rindex的作用与index基本相同，不过它是从右向左进行搜索。它的句法如下所示：

```
rindex string, substring
rindex string, substring, start_position
```

当搜索到结尾时，rindex返回-1。下面是一些例子：

```
$a="She loves you yeah, yeah, yeah.";
rindex($a, "yeah");        # Returns 26.
rindex($a, "yeah", 25);    # Returns 20
```

用于index的遍历循环与使用rindex进行向后搜索的循环略有不同。rindex的起点必须从字符的结尾开始，或者从结尾的后面开始，（在下例中，从length($source)开始），但是，当返回-1时，它仍然应该结束运行。当找到每个字符串后，$ start必须递减1，而不是像index那样递增1。

```
$source="One fish, two fish, red fish, blue fish.";
$start=length($source);
while(($start=rindex($source, "fish", $start)) != -1) {
    print "Found a fish at $start\n";
    $start--'
}
```

9.1.3 用substr分割标量

substr是个常常被忽略和很容易被遗忘的函数，不过它提供了一种从标量中取出信息并对标量进行编辑的通用方法。substr的句法如下：

```
substr string, offset
substr string, offset, length
```

substr函数取出string，从位置offset开始运行，并返回从offset到结尾的字符串的剩余部分。如果设定了length，那么取出length指明的字符，或者直到找出字符串的结尾，以先到者为准，如下例所示：

```
#Character positions in $a
#   0         10        20        30
$a="I do not like green eggs and ham.";
print substr($a, 25);      # prints "and ham."
print substr($a, 14, 5);   # prints "green"
```

如果offset设定为负值，substr函数将从右边开始计数。例如，substr（$a,-5）返回$a的最后5个字符。如果length设定为负值，则substr返回从它的起点到字符串结尾的值，少于 length指明的字符，如下例所示：

```
print substr($a, 5, -10);    # prints "not like green egg"
```

在上面这个代码段中，substr从位置5开始运行，返回字符串的剩余部分，但不包含最后10个字符。

你也可以使用赋值表达式左边的 substr函数。当用在左边时，substr用于指明标量中的什么字符将被替换。当用在赋值表达式的左边时，substr的第一个参数必须是个可以赋值的值，比如标量变量，而不应该是个字符串直接量。下面是使用 substr对字符串进行编辑的一个例子：

```
$a="countrymen, lend me your wallets";
# Replace first character of $a with "Romans, C"
substr($a, 0, 1)="Romans, C";

# Insert "Friends" at the beginning of $a
substr($a, 0, 0)="Friends, ";

substr($a, -7, 7)="ears.";          # Replace last 7 characters.
```

9.2 转换而不是替换

下一个运算符是转换运算符（有时称为翻译运算符），它使我们想起正则表达式中的替换的操作方式。替换操作符的形式是 s/pattern/reptacement/，在第6学时中我们已经作了介绍。除非你用连接运算符＝～设定了另一个标量，否则该操作符将对＄ _变量进行操作。转换操作符的作用与它有些类似，不过它并不使用正则表达式，而且它的运行方式完全不同。转换操作符的句法如下所示：

```
tr/searchlist/replacementlist/
```

转换操作符tr///用于搜索一个字符串，找出searchlist中的各个元素，并用replacementlist中的对应元素对它们进行替换。按照默认设置，转换操作符用于对变量＄ _进行搜索和修改。若要搜索和修改其他变量，你可以像使用正则表达式进行匹配操作那样，使用连接运算符，如下所示：

```
tr/ABC/XYZ/;          # In $_, replaces all A's with X's, B's with Y's, etc..
$r=~tr/ABC/XYZ/;      # Does the same, but with $r
```

字符的逻辑分组之间可以使用连字符。例如 A－Z代表大写字母A到Z，这样你就不必将它们全部写出来，请看下例：

```
tr/A-Z/a-z/;              # Change all uppercase to lowercase
tr/A-Za-z/a-zA-Z/;        # Invert upper and lowercase
```

如果replacementlist是空的，或者与searchlist相同，那么tr///将计算并返回匹配的字符。目

标字符串并不被修改，如下例所示：

```
$eyes=$potato=~tr/i//;      # Count the i's in $potato, return to $eyes
$nums=tr/0-9//;             # Count digits in $_, return to $nums
```

最后要说明的是，由于历史的原因， tr///也可以写成y///，其结果相同，因为y与tr同义。tr///运算符（和y///）也允许你为searchlist和replacementlist设定另一组界限符。这些界限符可以是任何一组自然配对的字符，如括号或任何其他字符，请看下面的例子：

```
tr(a-z)(n-za-m);          # Rotate all characters 13 to the left in $_
y[,._-][;:=|];            # Switch around some punctuation
```

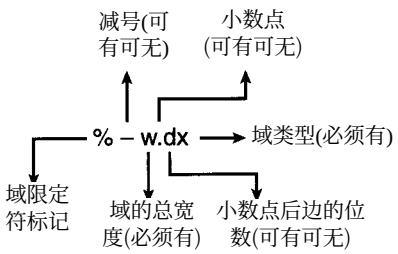
tr///运算符实际上还具备另外一些功能，不过用得不多。若要了解 tr///能够执行的所有其他任务，请查看perlop节中的在线文档。

9.3 功能更强的print函数

print函数是个非常简单的输出函数，它几乎不具备任何格式化功能。为了更具体地控制输出操作，如左对齐和右对齐，十进制精度，以及固定宽度的输出，你可以使用 Perl的printf函数。printf函数是从C编程语言那里借用的（几乎是原原本本的借用），不过其他编程语言也配有类似的函数，如BASIC的print using函数。printf函数的句法如下：

```
printf formatstring, list
printf filehandle formatstring, list
```

formatstring是一个描述输出格式的字符串，下面我们很快就要对它进行介绍。 list是一个你想让printf显示的值的列表，它类似 print语句中的list。通常而言， printf将它的输出显示给STDOUT文件句柄，但与print一样，如果你设定了一个文件句柄，那么 printf就使用该文件句柄。请注意， filehandle名与formatstring之间不使用逗号。



通常情况下formatstring是个字符串直接量，它也可以是一个用来描述输出格式的标量，formatstring中的每个字符均按其原义输出，但是以%开头的字符则属例外。%表示这是一个域说明符的开始。域说明符的格式是% -w.dx，其中w是域需要的总宽度， d是小数点左边的位数（对于数字来说）和字符串域允许的总宽度， x表示输出的是数据类型。 x说明符前面的连字符表示该域在 w字符中左对齐，否则它进行右对齐。只有%和x是不可少的。表9-1列出了一些不同类型的域说明符。

表9-1 Printf函数的部分域说明符列表

域 类 型	含 义
c	字符
s	字符串
d	十进制整数；截尾的小数
f	浮点数

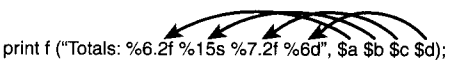
完整的域说明符列表请参见在线手册。你可以在命令提示符后面键入 perldoc -f printf，以查看该列表。

下面是使用printf的一些例子：

```
printf("%20s", "Jack");        # Right-justify "Jack" in 20-characters
printf("%-20s", "Jill");       # Left-justify "Jill" in 20 characters
$amt=7.12;
printf("%6.2f", $amt);         # prints "    7.12"
$amt=7.127;
printf("%6.2f", $amt);         # prints "    7.13", extra digits rounded
printf("%c", 65);              # prints ASCII character for 65, "A"
$amt=9.4;
printf("%6.2f", $amt);         # prints "   9.40"

printf("%6d", $amt);           # prints "      9"
```

每个格式说明符均使用列表中的一个项目，如上所示。对于每个项目来说，都应该有一个格式说明符；对于每个格式说明符来说，都有一个列表元素：



若要输出数字中的前导0，只需要在格式说明符中的宽度的前面设置 1个0，如下所示：

```
printf("%06.2f", $amt);  # prints "009.40"
```

sprintf函数与printf几乎相同，不过它不是输出值，而是输出 sprintf返回的格式化输出，你可以将它赋予一个标量，或者用于另一个表达式，如下所示：

```
$weight=85;
# Format result nicely to 2-decimals
$moonweight=sprintf("%.2f", $weight*.17);
print "You weigh $moonweight on the moon.";
```

请记住，带有%f格式说明符的printf和sprintf函数能够将计算结果圆整为你指定的小数点位数。

9.4 练习：格式化报表

当你使用计算机时，必然要处理的一项任务是将原始数据格式化为一个报表。计算机程序能够按照不同于人类阅读的格式对数据进行交换，常见的任务是使用该数据，并将它格式化为人能够阅读的报表。

为了进行这个练习，我们为你提供了一组员工记录，它包含关于某些虚构员工的信息，包括每小时的工资、工作的小时数、名字和员工号码。这个练习使用这些数据将它们重新格式化为一个很好的报表。

你可以很容易地修改这种类型的程序，以便输出其他类的报表。这个练习的数据包含在一个在程序开始初始化的数组中。在真实的报表中，数据可能来自磁盘上的一个文件。后面我们还要修改这个练习，以便使用外部的文件。

使用文本编辑器，键入程序清单 9-2中的程序，并将它保存为 Employee，不要键入行号。按照第1学时中的说明，使该程序成为可执行程序。

当完成上述操作后，在命令行提示符后面键入下面的命令，设法运行该程序。

```
Perl Employee
```

程序清单9-1显示了Employee程序的输出举例。

程序清单9-1 Employee程序的输出

```
1:   131211 Stan Cliburn          11.25   40   450.00
```

```
2:   132912 Alice Franklin        10.15  35  355.25
3:   141512 Wendy Ng               9.50  40  380.00
4:   123101 Bob Smith              9.35  40  374.00
5:   198131 Ted Wojohowicz         6.50  39  253.50
```

程序清单9-2 Employee程序的完整清单

```
1:  #!/usr/bin/perl -w
2:
3:  use strict;
4:
5:  my @employees=(
6:        'Smith,Bob,123101,9.35,40',
7:        'Franklin,Alice,132912,10.15,35',
8:        'Wojohowicz,Ted,198131,6.50,39',
9:        'Ng,Wendy,141512,9.50,40',
10:       'Cliburn,Stan,131211,11.25,40',
11: );
12:
13: sub print_emp {
14:      my($last,$first,$emp,$hourly,$time)=
15:              split(',', $_[0]);
16:      my $fullname;
17:      $fullname=sprintf("%s %s", $first, $last);
18:      printf("%6d %-20s %6.2f %3d %7.2f\n",
19:              $emp, $fullname, $hourly, $time,
20:              ($hourly * $time)+.005 );
21: }
22:
23: @employees=sort {
24:              my ($L1, $F1)=split(',', $a);
25:              my ($L2, $F2)=split(',', $b);
26:              return( $L1 cmp $L2  # Compare last names
27:                          ||   # If they're the same…
28:                      $F1 cmp $F2  # Compare first
29:              );
30:               } @employees;
31:
32: foreach(@employees) {
33:      print_emp($_);
34: }
```

第1行：这一行包含到达解释程序的路径（可以更改这个路径，使之适合系统的需要）和开关-w。请始终使警告特性处于激活状态。

第3行：use strict命令意味着所有变量都必须用my进行声明，裸单词必须用引号括起来。

第5~11行：员工列表被赋予@employees。数组中的每个元素均包含名字、姓、员工号、计时工资和工作的小时数。

第23~30行：@employees数组按姓和名字排序。

第24行：被排序的第1个元素（$a）分割为各个域。姓被赋予$L1，名字被赋于$F1。两者都用my声明为排序块的专用变量。

第25行：对另一个元素$b进行与上面相同的操作。姓名被赋于$L2和$L1。

第26~29行：使用类似第4学时中的程序清单4-1介绍的顺序，按字母对各个名字进行比较。

第32~34行：@employees中的已排序列表被传递给print-emp()，每次传递一个元素。

第13~21行：print-emp()函数输出格式化很好的员工记录。

第14~15行：传递来的记录$_[0]被分割成各个域，并被赋于变量$ last，$ first等，它们都是该子例程的专用变量。

第17行：名字和姓被合并为单个域，这样，两个域就可以放入某个宽度，并一道对齐。

第18~20行：记录被输出。$ hours与$ time相乘，得出合计金额。金额 .005与合计相加，这样，当乘积被截尾而成为两位数时，它能正确地圆整。

9.5 堆栈形式的列表

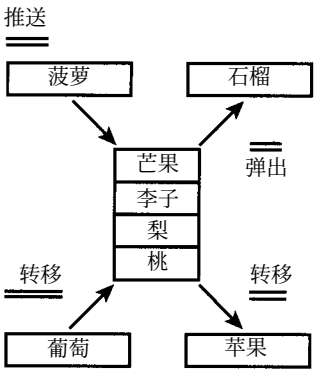
到现在为止，列表（和数组）一直是作为线性数据数组来展示的，其索引用于指明每个元素。

0	1	2	3	4	5
苹果	桃	梨	李子	芒果	石榴

请使用你的想象力，将各个组数元素设想为一个纵向堆栈。

在计算机术语中，这种列表称为堆栈。堆栈可用于按顺序来处理的累计操作。Klondike Solitaire游戏就是一个很好的例子。7堆牌中的每一堆牌分别代表一个堆栈；开始时，这些牌面向下放入堆栈。当需要这些牌时，它们被翻过来，并从堆栈中取走，再将其他牌放在新翻过来的牌的上面。

Perl中的堆栈通常用数组来实现。若要将各个项目放在堆栈的顶部，可以使用push函将项目压入堆栈；若要将项目从堆栈的顶部取出，可以使用 pop函数。



另外，堆栈可以从底部进行修改，可以将它视为从一叠纸牌的底部对它进行处理一样。Shift函数用于将元素添加到堆栈的底部， unshift用于从底部取出元素。每个函数的句法如下：

```
pop target_array

shift target_array

unshift target_array, new_list

push target_array, new_list
```

pop与shift函数分别从target_array中删除一个元素。如果target_array没有设定，那么元素可以从@_中删除，也可以从@ARGV中删除。pop和shift函数返回被删除的元素，如果数组是

空的，则返回undef。数组的大小将相应地缩小。

在子例程中，如果没有设定其他数组，那么 pop、shift、unshift和push函数将修改@_。在子例程外面，你的程序主体中，如果没有设定其他数组，那么这些函数将修改数组@ARGV。

push和unshift函数将new_list的元素添加给target_array，数组的大小将增大，以适应放置新元素的需要。被放入targer_array的项目，或者从target_array取出的项目既可以是一个列表，也可以是一个数组，如下例所示：

```
@band=qw(trombone);
push @band, qw( ukulele clarinet );
# @band now contains trombone, ukulele, clarinet

$brass=shift @band;     # $brass now has "trombone",
$wind=pop @band;        # $wind now has "clarinet"

# @band now contains only "ukulele"
unshift @band, "harmonica";
# @band now contains harmonica, ukulele
```

当你将元素添加给数组时，将元素推送（或移动）到数组中要比用手工将元素添加到数组的结尾处更加有效。比如， push(@list，@newitems)比@list=(@list，@newitems)更加有效。Perl的push、shift、unshift和pup函数都进行了优化，以适应这些操作的需要。

“堆栈”中的数组元素仍然属于标准的数组元素，可以用索引进行编址。堆栈的“底部”是元素0，堆栈的“顶部”是数组中的最后一个元素。

拼接数组

迄今为止，我们介绍了数组可以按元素进行寻址、分行、移动、弹出、取消移动和推送。数组操作的最后一个工具是splice。splice函数的句法如下：

```
splice array, offset
splice array, offset, length
splice array, offset, length, list
```

splice函数用于删除数组中从offset位置开始的元素，同时返回被删除的数组元素。如果offset的值是负值，则从数组的结尾处开始计数。如果设定了 length，那么只删除length指定的元素。如果设定了list，则删除length指定的元素，并用list的元素取代之。通过这个处理过程，数组就会根据需要扩大或缩小，如下面所示的那样：

```
@veg=qw( carrots corn );
splice(@veg, 0, 1);                          # @veg is corn
splice(@veg, 0, 0, qw(peas));                # @veg is peas, corn
splice(@veg, -1, 1, qw(barley, turnip));     # @veg is peas, barley, turnip
splice(@veg, 1, 1);                          # @veg is peas, turnip
```

9.6 课时小结

在本学时中，我们介绍了搜索其他字符串中的字符串时不需要使用正则表达式，你可以

使用index和rindex进行简单的搜索，也可以使用 tr///运算符进行简单的替换。 substr函数既可以用来从字符串中检索数据，也可以对它们进行编辑。你可以使用 printf和sprintf语句，用Perl创建格式很好的输出。另外，我们介绍了用作项目堆栈而不是平面列表的数组，还学习了如何对这些堆栈进行操作。

9.7 课外作业

9.7.1 专家答疑

问题：substr、index和rindex等函数真的有必要使用吗？当正则表达式可以用来执行它们的大多数操作时，为什么还要这几种函数？

解答：首先，用于进行简单的字符串搜索的正则表达式的运行速度比 index和rindex慢。第二，为字符位置固定的正则表达式编写替换表达式会产生很大的混乱，而有时 substr则是比较出色的解决方案。第三， Perl是一种丰富多彩的语言，你可以使用你喜欢的解决方案，你可以有多种多样的选择。

问题：如果我设定的索引位于标量的结尾之外，使用 substr（或index或rindex）将会出现什么情况？

解答：计算机的好处之一是：它具有很强的一致性，而且它有很强的耐心。对于“ 如果…将会出现什么情况 ”之类的问题，有时你只要试一试它的最容易实现的方法即可！什么是可能出现的最坏情况呢？

在这种情况下，如果你激活了警告特性，那么访问并不存在的标量的某个部分，就会产生一个“ use of undefined value ”（使用了未定义的值）的错误。例如，如果你使用 $ a=“ Foo ”；substr（$ a,5）；那么substr函数将返回undef。

9.7.2 思考题

1) 假如有下面这个代码，那么在 @A中将留下什么？

```
@A=qw(oats peas beans);
shift @A;
push @A, "barley";
pop;
```

a. oats peas beans

b. deans barley

c. peas beans barley

2) printf (“ %18.3f ”, $ a)这个代码能够进行什么操作？

a. 它输出一个浮点数，长度为 18个字符，小数点左边是 15个字符，小数点右边是 3个字符。

b. 它输出一个浮点数，小数点左边是 18个字符，小数点右边是3个字符。

c. 它输出一个浮点数，长度为18个字符，小数点左边是14个字符，右边3个字符。

3) 如果对一个字符串运行tr/a-z/A-Z，tr/A-Z/a-z能否使字符串恢复其原始形式？

a. 是，当然能够。

b. 也许做不到。

9.7.3 解答

1) 答案是c。shift删除了cats，而push则将barley添加到结尾处。最后的pop是个假像，它并没有设定任何数组，因此它从array @_中弹出某些数据，但是它没有给@A带来任何变化。

2) 答案是c。如果你猜测的答案是a，那么你没有将小数点计算在内，它在总数（18＝14＋1＋3）中占有一个位置。

3) 答案是b。tr/a-z/A-Z/转换的“ rosebud ”变成了“ ROSEBUD ”。如果试图用tr/A-Z/a-z/将它变回原样，那么产生的是“ rosebud ”，而不是原始字符串。

9.7.4 实习

- 使用标量而不是数组，重新编写第4学时中的Hangman游戏。你可以使用substr，对标量中的各个字符进行操作。
- 修改程序清单9-2，从文件中读取数据，而不是从数组中获取数据。打开文件，将数据读入一个数组，然后按正常情况继续操作。当然你必须在磁盘上创建该文件。