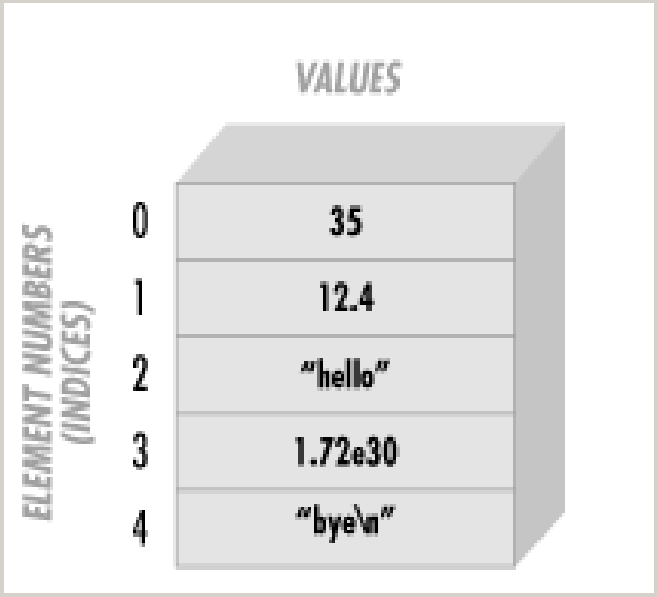
列表与数组的定义

- 列表是标量值的有序集，数组是用于存放列表的变量。每一个列表元素对应相应的数组元素，数组可以利用其索引查看每个元素的标量值。



列表的形式

- 列表元素包含在小括号()中，各元素用逗号隔开，可为数字、字符串、变量、表达式、undef值等同类型或不同类型的标量值。

```
()
(“hello”, ”world”)
print (1,2,3,4);   #输出1234
(‘a’, 1, 1.34, “hello, $name”)  #被替换为$name存放的直接量。
(1,2,3,)      #最后一个逗号被忽略
(1 .. 100)   #包含从1到100共100个元素
(1.7 .. 5.7)    #最小值和最大值换为整数
(“aa”..“ad”)   #等于("aa","ab","ac","ad")
($m .. $n)    # 由$m 和$n 的值决定
(1,$i+$j)     #表达式先计算结果，再输出
```

qw(quoted word)简写

- 利用qw可简化列表书写，其元素可以为数字、变量、不用引号的字符串、undef值，元素之间用空格分开
  格式：qw"界定符"元素1 元素2……元素n"界定符"。
  如：qw/fred barney betty wilma dino/
  qw/1 $a str baby/ #$a?

- 界定符可以选用任何标点符号，也可以使用成对的符号。
  如：qw #fred barney betty wilma dino#
  qw {fred barney betty wilma dino}
  qw (1 $a str baby)
  qw [1 $a str baby]

列表操作

- 列表为标量值的集合，因此可将其赋值给标量变量的集合

```
#! /usr/bin/perl
use strict;
my $fred;
my $barney;
my $dino;
($fred, $barney, $dino) = ("flintstone", "rubble", undef);
print "\$fred = $fred\n";
print "\$barney = $barney\n";
print "\$dino = $dino\n";
```

```
C:\Users\fqji>perl E:\test.pl
$fred = flintstone
$barney = rubble
$dino =
```

- 当左边列表中的变量数目 < 右边列表中的值数目，那么右边列表中多余的值将被抛弃
  ($first_name, $last_name) = qw(Jack Tom Brian);
  返回的结果为：
  $first_name="Jack"，$last_name="Tom"

- 当左边列表中的变量数目 > 右边列表中的值数目，那么左边列表中剩余的变量将被赋予undef值
  ($first_name, $middle_name, $last_name) = qw(Larry Wall);
  返回的结果为：
  $first_name = "Larry", $middle_name = "Wall",
  $last_name =undef

- 互换两个变量的值

```
#! /usr/bin/perl
use strict;
use warnings;
my $first_name = "Brian";
my $last_name = "Ruan";
print "CHANGE-before:\$first_name = $first_name\n";
print "CHANGE-before:\$last_name = $last_name\n";
($first_name, $last_name) = ($last_name, $first_name);
print "CHANGE-after:\$first_name = $first_name\n";
print "CHANGE-after:\$last_name = $last_name\n";
```

问题：与C语言的差别在哪？

```
C:\Users\fqji>perl E:\test.pl
CHANGE-before:$first_name = Brian
CHANGE-before:$last_name = Ruan
CHANGE-after:$first_name = Ruan
CHANGE-after:$last_name = Brian
```

数组形式

- 数组形式
  任何数组都是以@为标识，命名方式同简单变量
  例如：@a=(1,2,3);　　# @a与$a是不同的变量
  注意：数组的初始值为空()

- 元素形式
  数组的元素为简单变量，以$开头。后面的变量名同数组名，索引从0开始。
  例如：$a[0]，　$name[1]，$gene[3]

练习：定义核酸数组

```
C:\Users\fqji>perl E:\test.pl
Here are the array elements:
First element: A
Second element: C
Third element: G
Fourth element: T
```

定义核酸数组

```
# Here's one way to declare an array, initialized with a list of four scalar
    values.
@bases = ('A', 'C', 'G', 'T');

# Now we'll print each element of the array
print "Here are the array elements:";
print "\nFirst element: ";
print $bases[0];
print "\nSecond element: ";
print $bases[1];
print "\nThird element: ";
print $bases[2];
print "\nFourth element: ";
print $bases[3];
```

数组索引的一个例子

```
#! /usr/bin/perl
use warnings;
$number=2.71828;
@food;
for ($i=0; $i<5; $i++) {
$food[$i]=$i;
}
print "$food[$number+1]\n”
```

```
C:\Users\fqji>perl E:\test.pl
3
```

说明：任何求值能得到数字的表达式都可以用作下标

数组赋值方式

- @a=(1,2,3,4); #标准的列表赋值
- @a=(1..4); #@a为含元素1,2,3,4的数组
- @b=(1,2,3,4)，@a=@b; #数组复制， C 中可以这样吗？
- @b=(2,3)，@a=(1,@b,4); #数组内插 @a=(1,2,3,4)
- @b = (@odd, @even);
- @b=(‘a’) x 4; #$b=(‘a’, ‘a’, ‘a’, ‘a’)
- @b=<>; #从标准输入读入元素，用ctrl-Z结束输入

- @rocks = qw(bedrock $name lava);
  #返回的结果为：rocks[0]="bedrock"，rocks[1]='$name'（注意：不是变量$name的值），rocks[2]="lava“

```
C:\Users\fqji>perl
@rocks = qw( bedrock $name lava );
print "@rocks\n";
^Z
bedrock $name lava
```

- @city = (); @rocks = ("bedrock", @city, "lava");
  #数组rocks的元素为："bedrock"，"lava"，空列表被去除（隐示声名undef）

```
C:\Users\fqji>perl
@city=(); @rocks=( "bedrock", @city, "lava" );
print "@rocks\n";
^Z
bedrock lava
```

- @city = (undef); @rocks = ("bedrock", @city, "lava");
  #数组rocks的元素为："bedrock"，undef，"lava“

```
C:\Users\fqji>perl
@city=(undef); @rocks=( "bedrock", @city, "lava" );
print "@rocks\n";
^Z
bedrock  lava
```

- @rocks = ("bedrock", undef, "lava");
  #数组rocks的元素为："bedrock"，undef，"lava"（显示声明undef）

数组内插于字符串

- 数组可以插入双引号的字符串中，插入的数组元素会自动由空格。

```
@rocks = qw{ flintstone slate rubble };
print “quartz @rocks limestone\n”; #输出为5 种rocks,且由空格分开
@fred = qw(hello dolly);
$x =“This is $fred[1]’s place”; # “This is dolly’s place”
```

- 内插标量变量时，不要将其与[]连在一起，否则会被误认为是数组元素。

```
@fred = qw(eating rocks is wrong);
$fred = “right”;                    #期望打印“this is right[3]”
print “this is $fred[3]\n”;         #打印出“this is wrong”
print “this is ${fred}[3]\n”;       #使用花括号将变量隔离出来
print “this is $fred\[3]\n”;        #把接下来的字符转义
```

```
C:\Users\fqji>perl E:\test.pl
this is wrong
this is right[3]
this is right[3]
```

特殊的数组索引

- 如果将一个元素存储在数组最后元素之后的位置，数组会自动增长。

```
$rocks[0] = ‘bedrock’ ;
$rocks[1] = ‘slate ’ ;
$rocks[99] = ‘schist’ ;
# 除了以上三个元素，其他全为 undef 元素
```

- 针对rocks 数组，其最后一个元素的索引为$#rocks，即99。也可以用负索引值-1查看最后一个元素。

```
$rocks[$#rocks] = ‘schist’ ;
$rocks[-1] = ‘schist’ ;
```

数组元素读取

@a=(1..3)
单个数组元素读取：$a=$a[1];
多个数组元素读取：($x,$y,$z)=@a;
变量数多于数组元素：($a,$b,$c,$d)=@a; $a=1,$b=2,$c=3,$d=“”
变量数少于数组元素：($a,$b)=@a; $a=1,$b=2;

数组的输出

print @a; #输出的数组元素的值相连
print “@a”; #输出的数组元素的值之间加空格
print “@a\n”; #输出的数组的所有元素值后，再进行换行

数组切片

- 访问数组的部分元素：不是单元素，也不是全部
  例如：@a=(1,2,3,4,5)
  数组片断读出：@sub=@a[0,1,3] #@sub=(1,2,4)
  数组片断赋值：@a[1,3]=(“a”,”b”) #@a=(1,”a”,3,”b”,5)

- 数组切片的索引可用各种形式表示
  例如：@a=(1,2,3,4,5)
  范围：@sub=@a[1..3] #@sub=(2,3,4)
  $x=1,$y=3; @sub=@a[$x..$y]; #@sub=(2,3,4)
  @b=(1,2,3); @sub=@a[@b]; #@sub=(2,3,4)
  @b=(2,3); @sub=@a[1,@b]; @sub=(2,3,4)
  ($a,@b,$c)=(1,2,3,4,5) #@b=(2,3,4,5), $c=''

pop和push函数

- pop用来取出数组的最后一个元素并返回其值，同时该数组会减少一个元素。
  例如: @a=(1,2,3,4,5);
  $i=pop(@a);　#$i用于接收返回值5，现在@a是(1,2,3,4)
  注意：如果数组为空，pop什么也不做，返回undef

- 与pop相反，push向数组的最后添加一个或多个元素。
  @a=(1,2,3,4);
  push(@a,5);　#现在是(1,2,3,4,5)
  @b=(6,7);
  push(@a,@b); #现在是 (1,2,3,4,5,6,7)

shift和unshift函数

- push和pop对数组的末尾进行操作，unshift和shift对一个数组的开头进行操作

```
@array = qw/dino fred barney/;
$m = shift (@array);
#$m得到“dino”, @array 现在为(“fred”, “barney”)
$n = shift @array;
#$n得到”fred”, @array 现在为（“barney”）
shift @array; #@array 现在为空
$o = shift @array; #$o得到undef, @arry 仍为空
unshift(@array,5); #@array现在为(5)
unshift @array,4; #@array现在为(4,5)
@others = 1..3;
unshift @array, @others; #array现在为(1,2,3,4,5)
```

示例

```
#!/usr/bin/perl$
$
# Define an array$
@coins = ("Quarter","Dime","Nickel");$
print "First Statement : @coins";$
print "\n";$
$
# Add one element at the end of the array$
push(@coins, "Penny");$
print "Second Statement : @coins";$
print "\n";$
# Add one element at the beginning of the array$
unshift(@coins, "Dollar");$
print "Third Statement : @coins";$
print "\n";$
$
# Remove one element from the last of the array.$
pop(@coins);$
print "Fourth Statement : @coins";$
print "\n";$
# Remove one element from the beginning of the array.$
shift(@coins);$
print "Fifth Statement : @coins";$
```

```
C:\Users\fqji>perl E:\test.pl
First Statement : Quarter Dime Nickel
Second Statement : Quarter Dime Nickel Penny
Third Statement : Dollar Quarter Dime Nickel Penny
Fourth Statement : Dollar Quarter Dime Nickel
Fifth Statement : Quarter Dime Nickel
```

reverse和sort函数

- reverse（逆转）操作将输入的一串列表或数组按相反的顺序返回。

```
@fred = 6 ..10;
@barney = reverse (@fred); #得到10，9，8，7，6
```

- sort操作将列表排序，默认将列表的值按ASCII值进行升序排序。

```
@rocks = qw/bedrock slate rubble granite/;
@sorted = sort(@rocks); #得到bedrock, granite, rubble, slate
@back = reverse sort @rocks; #为slate 到bedrock
@array=(100,97,98,101,102,99)
@numbers = sort @ array; #得到100，101，102，97，98，99
@numbers =sort { $a<=>$b } @ array; #得到97，98，99，100，101，102
```