分子生物计算
(*Perl 语言编程*)

天津医科大学
生物医学工程与技术学院

2018-2019 学年上学期（秋）
2016 级生信班

第六章　子程序和 Bugs

伊现富（Yi Xianfu）

天津医科大学（TIJMU）
生物医学工程与技术学院

2018 年 12 月

教学提纲

教学提纲

子程序（subroutine）

- 对程序进行结构化组织的一个重要方法。
- 类似于 shell 编程语言中的函数。

Perl 调试器（debugger）

用“慢镜头”的形式来检查一个程序的行为，帮助找到 bugs。

子程序（subroutine）

- 对程序进行结构化组织的一个重要方法。
- 类似于 shell 编程语言中的函数。

Perl 调试器（debugger）

用“慢镜头”的形式来检查一个程序的行为，帮助找到 bugs。

教学提纲

教学提纲

子程序

- 子程序：把一些代码包裹起来，给它起一个名字，并提供方法把一些值传递给它进行计算，然后返回计算结果。
- 调用：程序的其余部分通过使用子程序的名字来使用其中的代码，把需要的值传递给它并收集运算结果。
- 程序中的程序：程序调用子程序得到结果就像你运行程序得到结果一样。
- 使用：只需知道传递哪些值（参数）、收集哪种类型的值（返回值）。
- 一次编写、多次使用：赋予程序抽象化和模块化的能力。

- 程序更加简短：在重用代码
- 更容易测试：单独对子程序进行测试
- 更容易理解：程序有良好的组织
- 更加稳健：代码量减少、出错几率变小
- 编写更加迅速：直接使用或者进行简单的“拼装”即可
- 程序更加灵活：程序可以不断增长但能适应各种情况
- 嵌套/递归：子程序可以调用其他的子程序（包括自己）

关键问题

如何把代码分割成一系列易于管理的子程序？

基本要求

- 子程序封装一些通用且有用的东西
- 编写的子程序不会只被调用一次

经验法则

- 子程序应该只做一件事情并把它做好（Unix 的基本原则之一）
- 子程序的代码最好不要超过一页或者两页

关键问题

如何把代码分割成一系列易于管理的子程序？

基本要求

- 子程序封装一些通用且有用的东西
- 编写的子程序不会只被调用一次

经验法则

- 子程序应该只做一件事情并把它做好（Unix 的基本原则之一）
- 子程序的代码最好不要超过一页或者两页

关键问题

如何把代码分割成一系列易于管理的子程序？

基本要求

- 子程序封装一些通用且有用的东西
- 编写的子程序不会只被调用一次

经验法则

- 子程序应该只做一件事情并把它做好（Unix 的基本原则之一）
- 子程序的代码最好不要超过一页或者两页

教学提纲

实例

- 要求：把“ACGT”拼接到指定 DNA 的末尾，返回新的、更长的 DNA
- 命名：addACGT
- 调用：子程序的名字后面跟上用小括号包裹起来的参数列表

```
1 addACGT($dna);
2 &addACGT($dna);
```

实例

- 要求：把“ACGT”拼接到指定 DNA 的末尾，返回新的、更长的 DNA
- 命名：addACGT
- 调用：子程序的名字后面跟上用小括号包裹起来的参数列表

```
1 addACGT($dna);
2 &addACGT($dna);
```

```
1  #!/usr/bin/perl -w
2  # Example 6-1   A program with a subroutine
     to append ACGT to DNA
3
4  # The original DNA
5  $dna = 'CGACGTCTTCTCAGGCGA';
6
7  # The call to the subroutine "addACGT".
8  # The argument being passed in is $dna; the
     result is saved in $longer_dna
9  $longer_dna = addACGT($dna);
10
11 print "I added ACGT to $dna and got
     $longer_dna\n\n";
12
13 exit;
```

```
15 ##########################################################
16 # Subroutines for Example 6-1
17 ##########################################################
18
19 # Here is the definition for subroutine "addACGT"
20
21 sub addACGT {
22     my ($dna) = @_;
23
24     $dna .= 'ACGT';
25     return $dna;
26 }
```

```
1 I added ACGT to CGACGTCTTCTCAGGCGA and got
    CGACGTCTTCTCAGGCGAACGT
```

程序分块

- 主程序/程序的主体（从开头到 exit 命令结束）
- 子程序的定义（剩余部分）

子程序的定义与调用

- 理论：放在程序的任何地方（使用它们的地方，程序的开头/末尾，散落各处）都是可以的
- 通常：集中放在程序的末尾（以字母顺序或者出现顺序等进行排列）
- 调用：子程序的名字后跟小括号包裹起来的参数（可以没有参数，多个参数要用逗号进行分隔）

程序分块

- 主程序/程序的主体（从开头到 exit 命令结束）
- 子程序的定义（剩余部分）

子程序的定义与调用

- 理论：放在程序的任何地方（使用它们的地方，程序的开头/末尾，散落各处）都是可以的
- 通常：集中放在程序的末尾（以字母顺序或者出现顺序等进行排列）
- 调用：子程序的名字后跟小括号包裹起来的参数（可以没有参数，多个参数要用逗号进行分隔）

```
1  #!/usr/bin/perl -w
2
3  $dna = 'CGACGTCTTCTCAGGCGA';
4
5  $longer_dna = addACGT($dna);
6
7  print "I added ACGT to $dna and got
     $longer_dna\n\n";
8
9  exit;
10
11 sub addACGT {
12     my ($dna) = @_;
13     $dna .= 'ACGT';
14     return $dna;
15 }
```

教学提纲

三部分

- 子程序定义的保留字 sub
- 子程序的名字（此处是 addACGT）
- 包裹在大括号中的代码块

```
1 sub addACGT {
2     my ($dna) = @_;
3     $dna .= 'ACGT';
4     return $dna;
5 }
```

两类变量

- 传递给子程序的参数
  - 参数：调用子程序时传递给它的值
  - 使用特殊变量 `@_` 向子程序传递参数值
- 子程序中声明的变量
  - 子程序使用的变量要与程序其他部分使用的变量区分开
  - 把这些变量的作用域（发挥作用的范围）限制在子程序中
  - 使用 my 声明变量

返回值

- 使用 return 函数返回子程序的结果
- 可以返回：标量、标量列表、数组，等

```
1 return $dna;
2 return ( $dna, $dna2 );
3 return @lines;
```

教学提纲

参数

- 参数（argument，parameter）通常包含子程序要计算的数据
- 调用时使用的参数名在子程序中无关紧要
- 关键的是被实际传递到子程序内部的参数的值
- 子程序从 `@_` 数组中收集参数的值，并把它们赋值给新的变量
- 新的变量名和调用时使用的变量名可以一样/不一样
- 参数值及值的顺序是不变的，而非变量名

```
1 # 注意：小括号表明是列表上下文，可以保证新变量能被正确
    地初始化
2 my ($dna) = @_;
3 my ($dna, $protein, $name_of_gene) = @_;
4 # 没有参数时，直接省略这样的语句即可
```

教学提纲

作用域与 my

- 作用域：把变量隐藏起来，使它们仅局限在程序的特定部分
- my（词法作用域）：把变量限制在使用它们的代码块中
- my 声明的变量可以和代码块外的变量重名

```
1  # 使用my声明变量
2  my ($x);
3  my $x;
4  # 声明变量的同时进行初始化
5  my $x = '49';
6
7  # 在子程序中收集参数
8  my ($x) = @_;
9  my $x = shift @_;
10 my $x = $_[0];
```

作用域与 my

- 作用域：把变量隐藏起来，使它们仅局限在程序的特定部分
- my（词法作用域）：把变量限制在使用它们的代码块中
- my 声明的变量可以和代码块外的变量重名

```
1  # 使用my声明变量
2  my ($x);
3  my $x;
4  # 声明变量的同时进行初始化
5  my $x = '49';
6
7  # 在子程序中收集参数
8  my ($x) = @_;
9  my $x = shift @_;
10 my $x = $_[0];
```

```
1  #!/usr/bin/perl -w
2  # Example 6-2   Illustrating the pitfalls of
     not using my variables
3
4  $dna = 'AAAAA';
5
6  $result = A_to_T($dna);
7
8  print "I changed all the A's in $dna to T's
     and got $result\n\n";
9
10 exit;
```

```
12 ##############################################################
13 # Subroutines
14 ##############################################################
15 sub A_to_T {
16     my ($input) = @_;
17
18     $dna = $input;
19
20     $dna =~ s/A/T/g;
21
22     return $dna;
23 }
```

预期输出

```
1 I changed all the A's in AAAAA to T's and got TTTTT
```

实际输出

```
1 I changed all the A's in TTTTT to T's and got TTTTT
```

修正子程序

```
1 sub A_to_T {
2     my ($input) = @_;
3     my $dna = $input;
4     $dna =~ s/A/T/g;
5     return $dna;
6 }
```

预期输出

```
1 I changed all the A's in AAAAA to T's and got TTTTT
```

实际输出

```
1 I changed all the A's in TTTTT to T's and got TTTTT
```

修正子程序

```
1 sub A_to_T {
2     my ($input) = @_;
3     my $dna = $input;
4     $dna =~ s/A/T/g;
5     return $dna;
6 }
```

预期输出

```
1 I changed all the A's in AAAAA to T's and got TTTTT
```

实际输出

```
1 I changed all the A's in TTTTT to T's and got TTTTT
```

修正子程序

```
1 sub A_to_T {
2     my ($input) = @_;
3     my $dna = $input;
4     $dna =~ s/A/T/g;
5     return $dna;
6 }
```

常见变量名

- **程序员常用：**`$tmp`, `$x`, `$a`, `$var`, `$array`, `$input`, `$output`, `$data`, `$result`, `$file`, ...
- **生物信息学家常用：**`$dna`, `$protein`, `$sequence`, `$motif`, ...
- 常见密码：123456, password, qwerty, abc123, 111111, iloveyou, admin, shadow, ...

强制使用 my 声明变量

```
1 use strict;
2 # 好处：谁用谁知道！
```

常见变量名

- **程序员常用：**`$tmp`, `$x`, `$a`, `$var`, `$array`, `$input`, `$output`, `$data`, `$result`, `$file`, ...
- **生物信息学家常用：**`$dna`, `$protein`, `$sequence`, `$motif`, ...
- 常见密码：123456, password, qwerty, abc123, 111111, iloveyou, admin, shadow, ...

强制使用 my 声明变量

```
1 use strict;
2 # 好处：谁用谁知道！
```

教学提纲

交互

- 人性化：与程序“面对面”，实时互动
- 需要用户实时值守

非交互

- 自动化：无人值守，计划任务
- 命令行界面

交互

- 人性化：与程序“面对面”，实时互动
- 需要用户实时值守

非交互

- 自动化：无人值守，计划任务
- 命令行界面

```
1  #!/usr/bin/perl -w
2  # Example 6-3   Counting the number of G's in
       some DNA on the command line
3
4  use strict;
5
6  # Collect the DNA from the arguments on the
     command line
7  #   when the user calls the program.
8  # If no arguments are given, print a USAGE
     statement and exit.
9
10 # $0 is a special variable that has the name
     of the program.
11 my ($USAGE) = "$0 DNA\n\n";
```

```
13 # @ARGV is an array containing all command-
     line arguments.
14 #
15 # If it is empty, the test will fail and the
     print USAGE and exit
16 #   statements will be called.
17 unless (@ARGV) {
18     print $USAGE;
19     exit;
20 }
```

```
22 # Read in the DNA from the argument on the
     command line.
23 my ($dna) = $ARGV[0];
24
25 # Call the subroutine that does the real work
     , and collect the result.
26 my ($num_of_Gs) = countG($dna);
27
28 # Report the result and exit.
29 print "\nThe DNA $dna has $num_of_Gs G\'s in
     it!\n\n";
30
31 exit;
```

```
37 sub countG {
38
39     # return a count of the number of G's in the
     argument $dna
40
41     # initialize arguments and variables
42     my ($dna) = @_;
43
44     my ($count) = 0;
45
46     # Use the fourth method of counting
     nucleotides in DNA, as shown in
47     # Chapter Four, "Motifs and Loops"
48     $count = ( $dna =~ tr/Gg// );
49
50     return $count;
51 }
```

```
1 AAGGGGTTTCCC
2
3 The DNA AAGGGGTTTCCC has 4 G's in it!
```

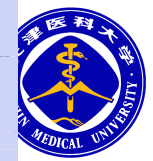
```
1 use strict;
```

作用

强制执行词法作用域（确保所有的变量都用 my 进行了声明）。

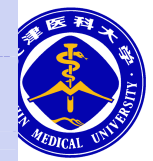
- `$0`：程序名
- `@ARGV`：所有的命令行参数

```
1 # $0 is a special variable that has the name
    of the program.
2 my ($USAGE) = "$0 DNA\n\n";
3
4 # @ARGV is an array containing all command-
    line arguments.
5 # If it is empty, the test will fail and the
    print USAGE and exit statements will be
    called.
6 unless (@ARGV) {
7     print $USAGE;
8     exit;
9 }
```



- 提示信息：程序名（$0）+ 程序需要的参数
- 步骤：检测参数，提示用户，退出程序

```
1 # $0 is a special variable that has the name
    of the program.
2 my ($USAGE) = "$0 DNA\n\n";
3
4 # @ARGV is an array containing all command-
    line arguments.
5 # If it is empty, the test will fail and the
    print USAGE and exit statements will be
    called.
6 unless (@ARGV) {
7     print $USAGE;
8     exit;
9 }
```



```
1 my ($dna) = $ARGV[0];
2 #my ($dna) = @ARGV;
3 #my $dna = $ARGV[0];
4 #my $dna = shift @ARGV;
```

说明

- 第一个元素的索引值是 0
- 提取元素时，把 @（表示数组）换成 $（表示标量）
- 下标要用中括号包裹起来

```
1  #!/usr/bin/perl -w
2  use strict;
3  my ($USAGE) = "$0 DNA\n\n";
4  unless (@ARGV) {
5      print $USAGE; exit;
6  }
7  my ($dna) = $ARGV[0];
8  my ($num_of_Gs) = countG($dna);
9  print "\nThe DNA $dna has $num_of_Gs G\'s in it!\n
     \n";
10 exit;
11
12 sub countG {
13     my ($dna) = @_;
14     my ($count) = 0;
15     $count = ( $dna =~ tr/Gg// );
16     return $count;
17 }
```

教学提纲

教学提纲

```
1  #!/usr/bin/perl -w
2  # Example of pass-by-value (a.k.a. call-by-value)
3
4  use strict;
5  my $i = 2;
6  simple_sub($i);
7  print "In main program, after the subroutine call,
       \$i equals $i\n\n";
8  exit;
9
10 sub simple_sub {
11   my($i) = @_;
12   $i += 100;
13   print "In subroutine simple_sub, \$i equals $i\n
     \n";
14 }
```

```
1 In subroutine simple_sub, $i equals 102
2
3 In main program, after the subroutine call,
    $i equals 2
```

```
1 simple_sub($i);
```

通过值（value）传递/调用

- 调用子程序时，参数的值被复制并传递给子程序；子程序中这些值的变化不会影响到主程序中相应参数的值。
- 适用于：传递单个标量、标量列表、单个数组

需求/问题

如果需要传递的参数比较复杂（混合标量、数组和散列）该怎么办呢？

```
1 simple_sub($i);
```

通过值（value）传递/调用

- 调用子程序时，参数的值被复制并传递给子程序；子程序中这些值的变化不会影响到主程序中相应参数的值。
- 适用于：传递单个标量、标量列表、单个数组

需求/问题

如果需要传递的参数比较复杂（混合标量、数组和散列）该怎么办呢？

教学提纲

```
1  #!/usr/bin/perl -w
2  # Example of problem of pass-by-value with two
     arrays
3  use strict;
4
5  my @i = ('1', '2', '3');
6  my @j = ('a', 'b', 'c');
7  print "In main program before calling subroutine:
     i = " . "@i\n";
8  print "In main program before calling subroutine:
     j = " . "@j\n";
9
10 reference_sub(@i, @j);
11 print "In main program after calling subroutine: i
     = " . "@i\n";
12 print "In main program after calling subroutine: j
     = " . "@j\n";
13 exit;
```

```
1 sub reference_sub {
2   my(@i, @j) = @_;
3
4   print "In subroutine : i = " . "@i\n";
5   print "In subroutine : j = " . "@j\n";
6
7   push(@i, '4');
8   shift(@j);
9 }
```

```
1 In main program before calling subroutine: i = 1 2 3
2 In main program before calling subroutine: j = a b c
3 In subroutine : i = 1 2 3 a b c
4 In subroutine : j =
5 In main program after calling subroutine: i = 1 2 3
6 In main program after calling subroutine: j = a b c
```

说明

- Perl 把 `@i` 和 `@j` 两个数组的所有元素都赋值给了子程序中的第一个数组 `@i`
- 使用词法作用域（即 my 变量），主程序中原始的数组不会被子程序所影响
- 解决办法：通过引用/参考/指针传递参数/调用子程序

```
1 reference_sub(\@i, \@j);
```

通过引用/参考/指针（reference）传递/调用

- 在变量名前加一个反斜线（\）
- 注意：在子程序中对参数变量值的操作会影响到主程序中参数的值
- 引用是存储在标量变量中的一种特殊类型的数据
- 从 @_ 数组中读取参数后要保存为标量变量
- 当使用引用时要对它们进行解引用
- 解引用：在引用前添加上表明变量类型的符号（标量 $，数组 @，散列 %）
- 解引用时在变量名前有两个符号：（从左到右）表明变量类型的本来的符号和表明是引用的 $ 符号（如：shift(@$j);）

```
1  #!/usr/bin/perl
2  # Example of pass-by-reference (a.k.a. call-by-reference
     )
3  use strict;
4  use warnings;
5
6  my @i = ('1', '2', '3');
7  my @j = ('a', 'b', 'c');
8  print "In main program before calling subroutine: i = "
     . "@i\n";
9  print "In main program before calling subroutine: j = "
     . "@j\n";
10
11 reference_sub(\@i, \@j);
12 print "In main program after calling subroutine: i = " .
     "@i\n";
13 print "In main program after calling subroutine: j = " .
     "@j\n";
14 exit;
```

```
1 sub reference_sub {
2   my($i, $j) = @_;
3
4   print "In subroutine : i = " . "@$i\n";
5   print "In subroutine : j = " . "@$j\n";
6
7   push(@$i, '4');
8   shift(@$j);
9 }
```

```
1 In main program before calling subroutine: i
    = 1 2 3
2 In main program before calling subroutine: j
    = a b c
3 In subroutine : i = 1 2 3
4 In subroutine : j = a b c
5 In main program after calling subroutine: i =
     1 2 3 4
6 In main program after calling subroutine: j =
     b c
```

```
1 # 通过值传递
2 simple_sub($i);
3
4 # 通过引用传递
5 reference_sub(\@i, \@j);
6
7 # 错误的传递
8 reference_sub(@i, @j);
```

教学提纲

模块（module）/库（library）

- 作用：避免繁琐、重复地复制粘贴子程序
- 把所有可重复使用的子程序统一放到一个或多个文件中
- 在程序中使用 use 函数把子程序的库文件读进来（就像它们本身就在程序中一样）
- 模块的后缀：`.pm`（比如：BeginPerlBioinfo.pm）
- 模块（`.pm` 文件）的最后一行必须是：`1;`
- 使用模块：在靠近程序顶部的地方加上语句 `use BiginPerlBioinfo;`（此处不需要 `.pm` 后缀）
- 必要时给出模块的全路径名

```
1 use lib '/home/tisdall/book';
2 use BeginPerlBioinfo;
```

教学提纲

bug

程序错误（Bug），或称漏洞，是程序设计中的术语，是指在软件运行中因为程序本身有错误而造成的功能不正常、死机、数据丢失、非正常中断等现象。

典故

1947 年 9 月 9 日，葛丽丝·霍普（Grace Hopper）发现了第一个电脑上的 bug。当在 Mark II 计算机上工作时，整个团队都搞不清楚为什么电脑不能正常运作了。经过大家的深度挖掘，发现原来是一只飞蛾意外飞入了一台电脑内部而引起的故障。这个团队把错误解除了，并在日记本中记录下了这一事件。也因此，人们逐渐开始用“Bug”（原意为“虫子”）来称呼计算机中的隐错。现在在华盛顿的美国国家历史博物馆中还可以看到这个遗稿。

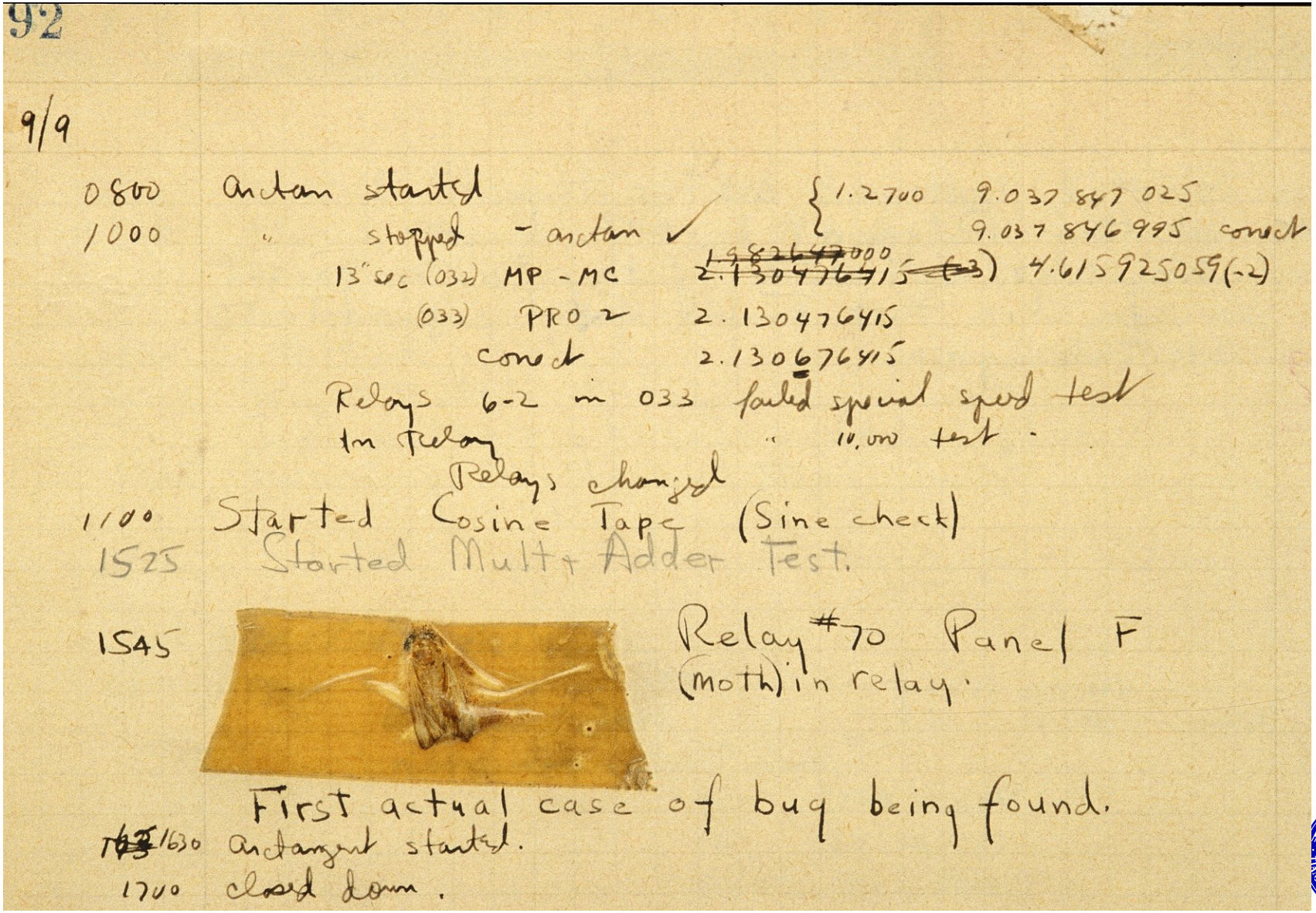
bug

程序错误（Bug），或称漏洞，是程序设计中的术语，是指在软件运行中因为程序本身有错误而造成的功能不正常、死机、数据丢失、非正常中断等现象。

典故

1947 年 9 月 9 日，葛丽丝·霍普（Grace Hopper）发现了第一个电脑上的 bug。当在 Mark II 计算机上工作时，整个团队都搞不清楚为什么电脑不能正常运作了。经过大家的深度挖掘，发现原来是一只飞蛾意外飞入了一台电脑内部而引起的故障。这个团队把错误解除了，并在日记本中记录下了这一事件。也因此，人们逐渐开始用“Bug”（原意为“虫子”）来称呼计算机中的隐错。现在在华盛顿的美国国家历史博物馆中还可以看到这个遗稿。

子程序和 Bugs | bug | 简介



Perl 脚本中常见的 bug

- 括号没配对
- 没用分号结尾
- 索引计算错误
- 变量/函数等拼写错误
- 本该用减法却用成了加法
- 意欲测试（==）却使用了赋值（=）
- 程序设计存在逻辑缺陷
- ……

教学提纲

use warnings;

```
1 #!/usr/bin/perl -w # 在Perl脚本中
2 use warnings;      # 在Perl脚本中
3 perl -w script.pl  # 在shell命令行上
```

开启 Perl 的警告功能，尝试寻找代码中潜在的问题（如：变量不止声明了一次），并给出警告。

use strict;

```
1 use strict;
```

- 强制声明变量（找到未声明的变量）
- 找到拼写错误的变量
- ……

use warnings;

```
1 #!/usr/bin/perl -w # 在Perl脚本中
2 use warnings;      # 在Perl脚本中
3 perl -w script.pl  # 在shell命令行上
```

开启 Perl 的警告功能，尝试寻找代码中潜在的问题（如：变量不止声明了一次），并给出警告。

use strict;

```
1 use strict;
```

- 强制声明变量（找到未声明的变量）
- 找到拼写错误的变量
- ……

教学提纲

选择性注释

- 通过不断的试验，发现当注释掉某部分代码时错误信息消失了，就知道是哪里出错了。
- 适用于没有精确定位错误位置、但是知道大体范围时。

添加 print 语句

- 添加 print 语句，打印出变量的值。
- 适用于差不多已经知道是哪儿出问题时。

选择性注释

- 通过不断的试验，发现当注释掉某部分代码时错误信息消失了，就知道是哪里出错了。
- 适用于没有精确定位错误位置、但是知道大体范围时。

添加 print 语句

- 添加 print 语句，打印出变量的值。
- 适用于差不多已经知道是哪儿出问题时。

教学提纲

程序功能设计

- 程序处理一条序列和两个碱基
- 如果能够在序列中找到这两个碱基的话，就把从这两个碱基到序列末尾的所有内容输出出来
- 可以以命令行参数的形式把这两个碱基传递给程序
- 如果不给参数，默认使用 TA 这两个碱基

```
1 #!/usr/bin/perl
2 # Example 6-4   A program with a bug or two
3 #
4 # An optional argument, for where to start
    printing the sequence,
5 #  is a two-base subsequence.
6 #
7 # Print everything from the subsequence ( or
    TA if no subsequence
8 # is given as an argument) to the end of the
    DNA.
```

```
10 # declare and initialize variables
11 my $dna = 'CGACGTCTTCTAAGGCGA';
12 my @dna;
13 my $receivingcommittment;
14 my $previousbase = '';
15
16 my $subsequence = '';
17
18 if (@ARGV) {
19     my $subsequence = $ARGV[0];
20 }
21 else {
22     $subsequence = 'TA';
23 }
```

```
25 my $base1 = substr( $subsequence, 0, 1 );
26 my $base2 = substr( $subsequence, 1, 1 );
27
28 # explode DNA
29 @dna = split( '', $dna );
30
31 ######## Pseudocode of the following loop:
32 #
33 # If you've received a committment, print the base
     and continue.  Otherwise:
34 #
35 # If the previous base was $base1, and this base
    is $base2, print them.
36 #   You have now received a committment to print
    the rest of the string.
37 #
38 # At each loop, save the previous base.
```

```
40 foreach (@dna) {
41     if ($receivingcommittment) {
42         print;
43         next;
44     }
45     elsif ( $previousbase eq $base1 ) {
46         if (/$base2/) {
47             print $base1, $base2;
48             $recievingcommitment = 1;
49         }
50     }
51     $previousbase = $_;
52 }
53
54 print "\n";
55
56 exit;
```

实际输出

```
1 $ perl example 6-4 AA
2
3 $ perl example 6-4
4 TA
```

理论输出

```
1 $ perl example 6-4 AA
2 AAGGCGA
3 $ perl example 6-4
4 TAAGGCGA
```

实际输出

```
1 $ perl example 6-4 AA
2
3 $ perl example 6-4
4 TA
```

理论输出

```
1 $ perl example 6-4 AA
2 AAGGCGA
3 $ perl example 6-4
4 TAAGGCGA
```

启动

```
1 # 交互式运行
2 perl -d script.pl   # 在shell命令行上
3
4 #自动启动
5 #!/usr/bin/perl -d # 在Perl脚本中
```

停止

在调试器中输入 q 即可。

启动

```
# 交互式运行
perl -d script.pl   # 在shell命令行上

#自动启动
#!/usr/bin/perl -d # 在Perl脚本中
```

停止

在调试器中输入 q 即可。

man perldebug Perl 调试器的联机帮助页

h 简短的帮助信息

h CMD 特定命令的帮助信息

h h 全部的帮助信息页

p print，打印出表达式/变量的值

n next，执行语句（把子程序看做单独的语句，直接跳过）

s single，执行语句（进入子程序，一步一步运行）

v view，查看临近的代码行

b breakpoint，（在指定行）设置断点

c continue，继续执行直到某个位置（比如：行，断点）

B 删除（某行的）一个断点或者所有断点（B *）

w watch，设置一个要查看/监视的表达式

R restart，尝试重新运行程序

- 调试器显示的是将要执行的那一行代码，而不是已经执行的代码行
- 使用 v 查看临近代码行时，当前行（即将被执行的行）会以 ==> 进行标示
- 重复键入 v 可以持续显示更多代码，使用减号 - 会上翻一屏
- 使用 print 打印数组时（`print @array`）默认元素之间没有空格，把数组放在双引号中（`print "@array"`）会使元素以空格分隔的形式展示出来
- 所谓断点（breakpoint）指的是程序中的一个点，调试器会在此处停止执行（避免从头开始一步一步执行代码中的每一行），便于检查附近的代码
- 特殊变量 `$_`：print 和模式匹配等默认使用的变量
- 特殊变量 `@_`：子程序存储参数的变量
- `use warings;` 和 `use strict;` 不是万能的，但强烈推荐同时使用它们两个
- 错误信息可能会有（一行）错位，所以真正的错误可能出现在提示行的前面

```
1  #!/usr/bin/perl
2  use warnings; use strict;
3
4  my $dna = "CGACGTCTTCTAAGGCGA";
5  my $subsequence = @ARGV ? $ARGV[0] : "TC";
6
7  my @dna = split "", $dna;
8  my $base1 = substr( $subsequence, 0, 1 );
9  my $base2 = substr( $subsequence, 1, 1 );
10
11 for ( my $i = 0 ; $i < @dna - 1 ; $i++ ) {
12     if ( $dna[$i] eq $base1 ) {
13         if ( $dna[ $i + 1 ] eq $base2 ) {
14             print join "", @dna[ $i .. $#dna ];
15             print "\n";
16             # last;
17         }
18     }
19 }
```

```
1  #!/usr/bin/perl
2
3  use warnings;
4  use strict;
5
6  my $dna = "CGACGTCTTCTAAGGCGA";
7  my $subsequence = @ARGV ? $ARGV[0] : "TC";
8
9  for ( my $i = 0 ; $i < length($dna) - 1 ; $i++ ) {
10     if ( substr( $dna, $i, 2 ) eq $subsequence ) {
11         print substr( $dna, $i );
12         print "\n";
13         # last;
14     }
15 }
```

```
1 my $subsequence = @ARGV ? $ARGV[0] : "TC";
```

```
1 my $subsequence;
2
3 if (@ARGV) {
4     $subsequence = $ARGV[0];
5 }
6 else {
7     $subsequence = "TC";
8 }
```

教学提纲

教学提纲

知识点

- 子程序：定义，调用，返回值，参数，作用域
- 命令行参数：特殊变量，提示信息，提取数组元素
- 传递数据给子程序：通过值，通过引用（引用与解引用）
- 模块：编写，使用，指定库目录
- 调试：use warnings; 和 use strict;，注释和 print 语句，调试器
- Perl 调试器：启动和停止，常用命令，设置断点

技能

- 能够熟练使用子程序
- 能够调试 Perl 程序
- 能够熟练使用 Perl 调试器

知识点

- 子程序：定义，调用，返回值，参数，作用域
- 命令行参数：特殊变量，提示信息，提取数组元素
- 传递数据给子程序：通过值，通过引用（引用与解引用）
- 模块：编写，使用，指定库目录
- 调试：use warnings; 和 use strict;，注释和 print 语句，调试器
- Perl 调试器：启动和停止，常用命令，设置断点

技能

- 能够熟练使用子程序
- 能够调试 Perl 程序
- 能够熟练使用 Perl 调试器

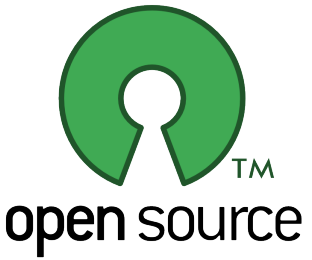
教学提纲

1. 如何定义和调用子程序？
2. 举例说明作用域的概念。
3. 如何获取命名行参数？
4. 给子程序传递数据的方法有哪些？举例说明。
5. 总结调试 Perl 程序的方法。
6. 如何使用 Perl 调试器对程序进行调试？

下节预告

回顾在 Linux 中有哪些方法可以实现随机化，或者随机选取一行/多行？（提示：sort，shuf）

Powered by



TEX LATEX XETEX Beamer