第7章　Perl语言编程

本章介绍有关Perl语言编程方面的内容。

7.1　什么是Perl

Perl(Practical Extraction and Report Language)是一种解释性的语言，专门为搜索纯文本文件而做了优化。它也可以十分方便地完成很多系统管理任务。它集成了 C、sed、awk和sh语言的优点，可以运行于Linux、UNIX、MVS、VMS、MS-DOS、Macintosh、OS/2、Amiga以及其他的一些操作系统上。特别是近年来，随着 Internet 的普及，Perl也越来越多地用于World Wide Web上CGI 等的编程，逐渐成为系统、数据库和用户之间的桥梁。

7.2　Perl的现状

大概有两种程序员喜欢用Perl：系统程序员可以用Perl结合系统命令一起处理数据和过程，并且可以使用Perl的格式匹配函数进行系统信息的搜寻和总结；还有一些开发 UNIX Web服务器CGI程序的程序员发现Perl比C语言易学易用，而且更加容易处理数据库和数据搜索。

Perl的创建人Larry Wall在1994年10月发表了Perl的第5版本。Perl 5增加了面向对象的能力，提供了更多的数据结构，与系统和数据库之间的新的标准接口以及其他的一些功能。

7.3　初试Perl

一个有用的Perl程序可以很短。例如我们希望更换大量文件中的一些相同内容，可以使用下面的一条命令：

```
perl -e 's/gopher/World Wide Web/gi' -p -i.bak *.html
```

下面是一个基本的perl程序：

```
#!/usr/local/bin/perl
#
# Program to do the obvious
#
print 'Hello world.';# Print a message
```

每个perl程序都以#!/usr/local/bin/perl开始，这样系统的外壳知道应该使用perl运行该程序。用#表示此后为注释语句。Perl的表达式必须以分号结尾，就如同C语言一样。此语句为显示语句，只是简单地显示出Hello world.字符串。

7.4　Perl变量

Perl中有三种变量：标量，数组（列表）和相关数组。

7.4.1　标量

Perl中最基本的变量类型是标量。标量既可以是数字，也可以是字符串，而且两者是可以互换的。具体是数字还是字符串，可以有上下文决定。标量变量的语法为 $variable_name。例

如：

```
$priority = 9;
```

把9赋予标量变量$priority，你也可以将字符串赋予该变量：

```
$priority = 'high';
```

注意 在Perl中，变量名的大小写是敏感的，所以$a 和$A是不同的变量。

以下的数值或字符串都可以赋给标量：

```
123  12.4  5E-10  0xff (hex)  0377 (octal)
'What you $see is (almost) what \n you get'   'Don\'t Walk'
"How are you?"  "Substitute values of $x and \n in \" quotes."
`date`  `uptime -u`  `du -sk $filespec | sort -n`
$x$list_of_things[5] $lookup{'key'}
```

从上面可以看出，Perl中有三种类型的引用。双引号（""）括起来的字符串中的任何标量和特殊意义的字符都将被Perl 解释。如果不想让Perl 解释字符串中的任何标量和特殊意义的字符，应该将字符串用单括号括起来。这时，Perl 不解释其中的任何字符，除了\\ 和\'。最后，可以用（`）将命令括起来，这样，其中的命令可以正常运行，并能得到命令的返回值。请看下面的例子：

```
1 #!/usr/bin/perl
2 $folks="100";
3 print "\$folks = $folks \n";
4 print '\$folks = $folks \n';
5 print "\n\n BEEP! \a \LSOME BLANK \ELINES HERE \n\n";
6 $date = `date +%D`;
7 print "Today is [$date] \n";
8 chop $date;
9 print "Date after chopping off carriage return: [".$date."]\n";
```

注意 实际程序中不应该包括行号。

其输出结果如下：

```
$folks = 100
$folks = $folks \n

BEEP! some blank LINES HERE

Today is [03/29/96]
Date after chopping off carriage return: [03/29/96]
```

第3行显示$folks的值。$之前必须使用换码符\，以便Perl显示字符串$folks而不是$folks的值100。

第4行使用的是单引号，结果Perl不解释其中的任何内容，只是原封不动地将字符串显示出来。

第6行使用的是（`），则date +%D命令的执行结果存储在标量$date中。

上例中使用了一些有特殊意义的字符，下面列出这些字符的含义：

\n	换行。
\r	回车。
\t	制表符。
\a	蜂鸣声。

\b	Backspace。
\L \E	将 \L 和\E之间的字符转换成小写。
\l	将其后的字符转换成小写 。
\U \E	将 \U 和\E 之间的字符转换成大写 。
\u	将其后的字符转换成大写 。
\cC	插入控制字符 C。
\x##	十六进制数 ## 。
\0ooo	八进制数ooo。
\\	反斜杠。
\	按原样输出下一个字符，例如： \$ 输出 $。

Perl中的数字是以浮点形式存储的。下面列出有关的数字运算符：

$a = 1 + 2;#	1加2，结果存储在$a中。
$a = 3 - 4;#	3减4，结果存储在 $a中。
$a = 5 * 6;#	5乘6，结果存储在 $a中。
$a = 7 / 8;#	7 除以8，结果存储在$a中。
$a = 9 ** 10;#	9的10次方，结果存储在$a中。
$a = 5 % 2;#	取5除2的余数，结果存储在$a中。
++$a;# $	a 加1，然后赋予$a。
$a++;#	先将$a返回，然后$a加1。
--$a;# $	a 减1，然后赋予$a。
$a--;#	先将$a返回，然后$a减1。

Perl 支持的逻辑运算符：

$r = $x || $y	$r = $x 或 $y。
$r = $x && $y	$r = $x 与 $y。
$r = ! $x	$r = 非 $x。

对于字符标量， Perl支持下面的运算符：

$a = $b . $c;#	将 $b 和$c连接，然后赋予$a。
$a = $b x $c;#	$b 重复$c 次，然后赋予$a。

下面是Perl 中的赋值方法：

$a = $b;#	将 $b 赋予 $a。
$a += $b;#	$b 加 $a，然后赋予$a。
$a -= $b;#	$a减 $b，然后赋予$a。
$a .= $b;#	把$b 连接到$a的后面，然后赋予$a。

你也可以使用下面的比较运算符：

$x == $y	如果$x 和$y相等，则返回真。
$x != $y	如果$x 和$y不相等，则返回真。
$x < $y	如果$x 比$y小，则返回真。
$x <= $y	如果$x小于等于$y，则返回真。
$x > $y	如果$x 比$y大，则返回真。
$x >= $y	如果$x 大于等于$y，则返回真。
$x eq $y	如果字符串$x 和字符串$y相同，则返回真。

$x ne $y	如果字符串$x 和字符串$y不相同，则返回真。
$x lt $y	如果字符串$x 比字符串$y小，则返回真。
$x le $y	如果字符串$x小于等于字符串$y，则返回真。
$x gt $y	如果字符串$x 比字符串$y大，则返回真。
$x ge $y	如果字符串$x 大于等于字符串$y，则返回真。
$x cmp $y	如果$x 大于 $y，则返回1，如果$x 等于 $y，则返回0，如果$x 小于$y，则返回-1。
$w ? $x : $y	如果$w为真，则返回 $x ；如果$w为假，则返回$y。
($x..$y)	返回从$x 到 $y之间的值。

7.4.2 数组

数组也叫做列表，是由一系列的标量组成的。数组变量以 @开头。请看以下的赋值语句：

```
@food  = ("apples",  "pears",  "eels");
@music = ("whistle",  "flute");
```

数组的下标从0开始，你可以使用方括号引用数组的下标：

```
$food[2]
```

返回eels。 注意@ 已经变成了 $ ， 因为eels 是一个标量。

在Perl中，数组有多种赋值方法，例如：

```
@moremusic = ("organ",  @music,  "harp");
@moremusic = ("organ",  "whistle",  "flute",  "harp");
```

还有一种方法可以将新的元素增加到数组中：

```
push(@food,  "eggs");
```

把eggs 增加到数组@food的末端。要往数组中增加多个元素，可以使用下面的语句：

```
push(@food,  "eggs",  "lard");
push(@food,  ("eggs",  "lard"));
push(@food,  @morefood);
```

push 返回数组的新长度。

pop用来将数组的最后一个元素删除，并返回该元素。例如：

```
@food  = ("apples",  "pears",  "eels");
$grub = pop(@food);#此时 $grub = "eels"
```

请看下面的例子：

```
1 #!/usr/bin/perl
2 #
3 # An example to show how arrays work in Perl
4 #
5 @amounts = (10,  24,  39);
6 @parts = ('computer',  'rat',  "kbd");
7
8 $a = 1; $b = 2; $c = '3';
9 @count = ($a,  $b,  $c);
10
11 @empty = ();
12
13 @spare = @parts;
14
15 print '@amounts = ';
```

```
16 print "@amounts \n";
17
18 print '@parts = ';
19 print "@parts \n";
20
21 print '@count = ';
22 print "@count \n";
23
24 print '@empty = ';
25 print "@empty \n";
26
27 print '@spare = ';
28 print "@spare \n";
29
30
31 #
32 # Accessing individual items in an array
33 #
34 print '$amounts[0] = ';
35 print "$amounts[0] \n";
36 print '$amounts[1] = ';
37 print "$amounts[1] \n";
38 print '$amounts[2] = ';
39 print "$amounts[2] \n";
40 print '$amounts[3] = ';
41 print "$amounts[3] \n";
42
43 print "Items in \@amounts = $#amounts \n";
44 $size = @amounts; print "Size of Amount = $size\n";
45 print "Item 0 in \@amounts = $amounts[$[]\n";
```

以下是显示结果：

```
@amounts = 10 24 39
@parts = computer rat kbd
@count = 1 2 3
@empty =
@spare = computer rat kbd
$amounts[0] = 10
$amounts[1] = 24
$amounts[2] = 39
$amounts[3] =
Items in @amounts = 2
Size of Amount = 3
Item 0 in @amounts = 10
```

第5行，三个整数值赋给了数组 @amounts。第6行，三个字符串赋给了数组@parts。第8行，字符串和数字分别赋给了三个变量，然后将三个变量赋给了数组 @count。11行创建了一个空数组。13行将数组 @spare赋给了数组@parts。

15到28行输出了显示的前5行。34到41行分别存取数组@amounts的每个元素。

注意 $amount[3]不存在，所以显示一个空项。

43行中使用$#array方式显示一个数组的最后一个下标，所以数组 @amounts的大小是($#amounts + 1)。44行中将一个数组赋给了一个标量，则将数组的大小赋给了标量。 45行使用了一个Perl中的特殊变量$[，用来表示一个数组的起始位置（缺省为0）。

7.4.3 相关数组

一般的数组允许我们通过数字下标存取其中的元素。例如数组 @food的第一个元素是$food[0]，第二个元素是$food[1]，以此类推。但Perl允许创建相关数组，这样我们可以通过字符串存取数组。其实，一个相关数组中每个下标索引对应两个条目，第一个条目叫做关键字，第二个条目叫做数值。这样，你就可以通过关键字来读取数值。

相关数组名以百分号(%)开头，通过花括号({})引用条目。例如：

```
%ages = ("Michael Caine", 39,
"Dirty Den", 34,
"Angie", 27,
"Willy", "21 in dog years",
"The Queen Mother", 108);
```

这样我们可以通过下面的方法读取数组的值：

```
$ages{"Michael Caine"};# Returns 39
$ages{"Dirty Den"};# Returns 34
$ages{"Angie"};# Returns 27
$ages{"Willy"};# Returns "21 in dog years"
$ages{"The Queen Mother"};# Returns 108
```

7.5 文件句柄和文件操作

我们可以通过下面的程序了解一下文件句柄的基本用法。此程序的执行结果和 UNIX系统的cat命令一样：

```
#!/usr/local/bin/perl
#
# Program to open the password file,  read it in,
# print it,  and close it again.
$file = '/etc/passwd';# Name the file
open(INFO,  $file);# Open the file
@lines = <INFO>;# Read it into an array
close(INFO);# Close the file
print @lines;# Print the array
```

open函数打开一个文件以供读取，其中第一个参数是文件句柄（ filehandle）。文件句柄用来供Perl在以后指向该文件。第二个参数指向该文件的文件名。 close函数关闭该文件。

open函数还可以用来打开文件以供写入和追加，只须分别在文件名之前加上 >和>>：

```
open(INFO,  $file);# Open for input
open(INFO,  ">$file");# Open for output
open(INFO,  ">>$file");# Open for appending
open(INFO,  "<$file");# Also open for input
```

另外，如果你希望输出内容到一个已经打开供写入的文件中，你可以使用带有额外参数的print语句。例如：

```
print INFO "This line goes to the file.\n";
```

最后，可以使用如下的语句打开标准输入（通常为键盘）和标准输出（通常为显示器）：

```
open(INFO, '-');# Open standard input
open(INFO, '>-');# Open standard output
```

一个Perl 程序在它一启动时就已经有了三个文件句柄：STDIN (标准输入设备)， STDOUT (标准输出设备) 和STDERR (标准错误信息输出设备)。

如果想要从一个已经打开的文件句柄中读取信息，可以使用 < > 运算符。

使用read 和write 函数可以读写一个二进制的文件。其用法如下：

```
read(HANDLE, $buffer, $length[, $offset]);
```

此命令可以把文件句柄是HANDLE的文件从文件开始位移$offset处，共$length字节，读到$buffer中。其中$offset是可选项，如果省略$offset，则read()从当前位置的前$length个字节读取到当前位置。可以使用下面的命令查看是否到了文件末尾：

```
eof(HANDLE);
```

如果返回一个非零值，则说明已经到达文件的末尾。

打开文件时可能出错，所以可以使用 die()显示出错信息。下面打开一个叫做“ test.data ”的文件：

```
open(TESTFILE, "test.data") || die "\n $0 Cannot open $! \n";
```

7.6 循环结构

7.6.1 foreach循环

在Perl 中，可以使用foreach循环来读取数组或其他类似列表结构中的每一行。请看下面的例子：

```
foreach $morsel (@food)# Visit each item in turn
# and call it $morsel
{
print "$morsel\n";# Print the item
print "Yum yum\n";# That was nice
}
```

每次要执行的命令用花括号括出。第一次执行时 $morsel被赋予数组@food的第一个元素的值，第二次执行时$morsel被赋予数组@food的第二个元素的值，以此类推直到数组的最后一个元素。

7.6.2 判断运算

在Perl中任何非零的数字和非空的字符串都被认为是真。零、全为零的字符串和空字符串都为假。

下面是一些判断运算符：

$a == $b　　如果$a 和$b相等，则返回真。

$a != $b　　如果$a和$b不相等，则返回真。

$a eq $b　　如果字符串$a和字符串$b相同，则返回真

$a ne $b　　如果字符串$a和字符串$b不相同，则返回真。

你可以使用逻辑运算符：

($a && $b)　$a与$b。

($a || $b)$a　或$b。

!($a)　　非$a。

7.6.3 for循环

Perl 中的 for 结构和C语言中的for 结构基本一样：

```
for (initialise; test; inc){
 first_action;
 second_action;
 etc
}
```

下面是一个for 循环的例子，用来显示从0到9的数字：

```
for ($i = 0; $i < 10; ++$i)# Start with $i = 1
        # Do it while $i < 10
# Increment $i before repeating
{
print "$i\n";
}
```

7.6.4 while 和until循环

下面是一个while 和until循环的例子。它从键盘读取输入直到得到正确的口令为止。

```
#!/usr/local/bin/perl
print "Password? ";# Ask for input
$a = <STDIN>;# Get input
chop $a;# Remove the newline at end
while ($a ne "fred")# While input is wrong...
{
print "sorry. Again? ";# Ask again
$a = <STDIN>;# Get input again
chop $a;# Chop off newline again
}
```

当输入和口令不相等时，执行while 循环。

你也可以在执行体的末尾处使用while 和until ， 这时需要用do语句：

```
#!/usr/local/bin/perl
do
{
"Password? ";# Ask for input
$a = <STDIN>;# Get input
chop $a;# Chop off newline
}
while ($a ne "fred")# Redo while wrong input
```

7.7 条件结构

Perl 也允许 if/then/else 表达式。请看下面的例子：

```
if ($a) {
print "The string is not empty\n";
}
else {
print "The string is empty\n";
}
```

注意 在Perl中，空字符被认为是假。

If结构中也可以使用嵌套结构：

```
if (!$a)# The ! is the not operator
{
print "The string is empty\n";
}
elsif (length($a) == 1)# If above fails，try this
{
print "The string has one character\n";
}
elsif (length($a) == 2)# If that fails，try this
{
print "The string has two characters\n";
}
else# Now，everything has failed
{
print "The string has lots of characters\n";
}
```

7.8 字符匹配

Perl 字符匹配功能十分强大。字符匹配功能的核心是规则表达式（ RE），也就是字符匹配过程中涉及到的格式。=~运算符用来进行格式匹配和替换。例如：

如果：

```
$s = 'One if by land and two if by sea';
```

则：

```
if ($s =~ /if by la/) {print "YES"}
else {print"NO"}
```

将会显示 YES，因为if by la在字符串$s中。再例如：

```
if ($s =~ /one/) {print "YES"}
else {print "NO"}
```

将显示 NO，因为RE是对大小写敏感的。如果使用 i选项，则将忽略大小写，则下面会显示出YES：

```
if ($s =~ /one/i) {print "YES"}
else {print "NO"}
```

下面列出了RE中许多具有特殊意义的字符：

. 任何字符除了换行符（\n）

^ 一行和一个字符串的开始

$ 一行和一个字符串的结束

* 其前一个字符重复零次或多次

+ 其前一个字符重复一次或多次

? 其前一个字符重复零次或一次

例如：

```
if ($x =~ /l.mp/) {print "YES"}
```

对于$x = “ lamp ”、“ lump ”、“ slumped ”将显示 YES，但对于$x = “ lmp ”或“ less amperes ”将不会显示YES。

再看下面的例子：

/fr.*nd/ 匹配 frnd、friend、front and back。
/fr.+nd/ 匹配 frond、friend、front and back。但不匹配frnd。
/10*1/ 匹配 11、101、1001、100000001。
/b(an)*a/ 匹配 ba、 bana、banana、bananananana。
/flo?at/ 匹配 flat和float，但不匹配flooat。

方括号用来匹配其中的任何字符。方括号中的 -符号用来表明在什么之间， ^符号表明非的意思。

[0123456789] 匹配任何单个的数字。
[0-9] 匹配任何单个的数字。
[a-z]+ 匹配任何由小写字母组成的单词。
[^0-9] 匹配任何非数字的字符。

反斜杠还是用于转义。如果你想匹配+、?、.、*、^、$、(、)、[、]、{、}、|、\ 和/这些字符，则其前面必须用反斜杠（\）。例如：

/10.2/ 匹配 10Q2、1052和10.2。
/10\.2/ 匹配 10.2，但不和10Q2或 1052匹配。
/*+/ 匹配一个或多个星号。
/A:\\DIR/ 匹配 A:\DIR。
/\/usr\/bin/ 匹配 /usr/bin。

下面还有一些特殊意义的字符：

\n 换行。
\t 制表符。
\w 任何字母和数字和[a-zA-Z0-9_]一样。
\W 任何非字母和数字和[^a-zA-Z0-9_]一样。
\d 任何数字和[0-9]一样。
\D 任何非数字和[^0-9]一样。
\s 任何空白字符：空格、 tab、换行等。
\S 任何非空白字符。
\b 单词边界，只对 [] 以外有效。
\B 非单词边界。

7.9 替换和翻译

7.9.1 替换

Perl可以使用s函数利用字符匹配的结果进行字符替换。 s 函数和vi编辑器的作用基本一样。这时还是使用字符匹配运算符=~，例如：

将字符串$sentence中所出现的london 用London 替换，可以使用如下的命令：

```
$sentence =~ s/london/London/
```

命令的返回值是所做的替换数目。

但此命令只能替换第一个出现的 london。如果希望将所有在字符串中出现的 london都用London替换，则应使用/g选项：

```
s/london/London/g
```

此命令的对象是$_变量，也就是当前的缺省变量。

如果希望还能替换类似lOndon、lonDON、LoNDoN的字符串，可以使用：

```
s/[Ll][Oo][Nn][Dd][Oo][Nn]/London/g
```

但更为简单的方法是使用i选项，也就是忽略大小写：

```
s/london/London/gi
```

7.9.2 翻译

tr函数允许逐字地翻译。下面的命令使得字符串$sentence中的a、b、c分别由e、f、d代替：

```
$sentence =~ tr/abc/efd/
```

结果返回所做的替换数目。

大多数RE中的特殊字符在tr函数中并不存在。例如下面的命令用来计算字符串 $sentence 中星号（*）的数目，并将结果存储在$count:

```
$count = ($sentence =~ tr/*/*/);
```

7.10 子过程

7.10.1 子过程的定义

Perl语言也可以定义自己的子过程。子过程的定义如下：

```
sub mysubroutine{
print "Not a very interesting routine\n";
print "This does the same thing every time\n";
}
```

下面的几种方法都可以调用这个子过程：

```
&mysubroutine;# Call the subroutine
&mysubroutine($_);# Call it with a parameter
&mysubroutine(1+2, $_);# Call it with two parameters
```

7.10.2 参数

调用一个子过程时，所有的参数都传送到了数组 @_中。下面子过程的例子显示出所有的参数：

```
sub printargs{
print "@_\n";
}
&printargs("perly", "king");# Example prints "perly king"
&printargs("frog", "and", "toad"); # Prints "frog and toad"
```

7.10.3 返回值

下面的例子返回两个输入参数的最大值：

```
sub maximum{
if ($_[0] > $_[1]){
$_[0];
}
else{
$_[1];
```

```
}
}
$biggest = &maximum(37, 24);# Now $biggest is 37
```

7.11 Per程序的完整例子

最后请看一个Perl语言的完整的例子。

此程序从一个记录学生信息的文件 stufile 和一个记录学生成绩的文件 scorefile 中生成一个学生成绩报告单。

输入文件stufile 由学生ID、姓名和年级三个字段组成，其间由分号隔开：

```
123456；Washington, George；SR
246802；Lincoln, Abraham "Abe"；SO
357913；Jefferson, Thomas；JR
212121；Roosevelt, Theodore "Teddy"；SO
```

文件scorefile 由学生ID、科目号、分数三个字段组成，由空格隔开：

```
123456 1 98
212121 1 86
246802 1 89
357913 1 90
123456 2 96
212121 2 88
357913 2 92
123456 3 97
212121 3 96
246802 3 95
357913 3 94
```

程序应该输出如下的结果：

```
Stu-ID Name...1 2 3 Totals:
357913 Jefferson, Thomas90 92 94 276
246802 Lincoln, Abraham "Abe"89 95 184
212121 Roosevelt, Theodore "Teddy"86 88 96 270
123456 Washington, George98 96 97 291
 Totals: 363 276 382
```

源程序如下：

```
#!/usr/local/bin/perl
# Gradebook - demonstrates I/O, associative
# arrays, sorting, and report formatting.
# This accommodates any number of exams and students
# and missing data. Input files are:
$stufile='stufile';
$scorefile='scorefile';
# If file opens successfully, this evaluates as "true", and Perl
# does not evaluate rest of the "or" "||"
open (NAMES, "<$stufile") || die "Can't open $stufile $!";
open (SCORES, "<$scorefile") || die "Can't open $scorefile $!";
# Build an associative array of student info
# keyed by student number
while (<NAMES>) {
($stuid, $name, $year) = split(':', $_);
```

```
$name{$stuid}=$name;
if (length($name)>$maxnamelength) {
$maxnamelength=length($name);
}
}
close NAMES;
# Build a table from the test scores:
while (<SCORES>) {
($stuid,  $examno,  $score) = split;
$score{$stuid,  $examno} = $score;
if ($examno > $maxexamno) {
$maxexamno = $examno;
}
}
close SCORES;
# Print the report from accumulated data!
printf "%6s %-${maxnamelength}s ",
'Stu-ID',  'Name...';
foreach $examno (1..$maxexamno) {
printf "%4d",  $examno;
}
printf "%10s\n\n",  'Totals:';
# Subroutine "byname" is used to sort the %name array.
# The "sort" function gives variables $a and $b to
# subroutines it calls.
# "x cmp y" function returns -1 if x<y,   0 if x=y,
# +1 if x>y.  See the Perl documentation for details.
sub byname { $name{$a} cmp $name{$b} }
# Order student IDs so the names appear alphabetically:
foreach $stuid ( sort byname keys(%name) ) {
# Print scores for a student,   and a total:
printf "%6d %-${maxnamelength}s ",
$stuid,  $name{$stuid};
$total = 0;
foreach $examno (1..$maxexamno) {
printf "%4s",  $score{$stuid,  $examno};
$total += $score{$stuid,  $examno};
$examtot{$examno} += $score{$stuid,  $examno};
}
printf "%10d\n",  $total;
}
printf "\n%6s %${maxnamelength}s ",  "Totals: ";
foreach $examno (1..$maxexamno) {
printf "%4d",  $examtot{$examno};
}
print "\n";
exit(0);
```